TELECOMUNICACIÓN

Campus Sur

POLITÉCNICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: UNIDAD DE EFECTOS DE AUDIO EN SIMULINK

AUTOR: JAVIER CARCELLER VARONA

TITULACIÓN: IMAGEN Y SONIDO

TUTOR:

ANTONIO MÍNGUEZ OLIVARES

DEPARTAMENTO:

DEPARTAMENTO DE INGENIERÍA AUDIOVISUAL Y DE COMUNICACIONES

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: NICOLÁS LÓPEZ PÉREZ

VOCAL: ANTONIO MÍNGUEZ OLIVARES

SECRETARIO: FRANCISCO JAVIER TABERNERO GIL

Fecha de lectura:

Calificación:

El Secretario,

A mis padres, por todo su apoyo.

"Sin música, la vida sería un error" - Friedrich Wilhelm Nietzsche

AGRADECIMIENTOS

Quiero dedicar este libro fruto de un año completo de duro trabajo a mis padres. Ellos han sido y son, sin duda, el gran apoyo que me ha permitido llegar hasta aquí. Gracias a su duro esfuerzo, sobre todo en estos tiempos aciagos que corren, su comprensión y su cariño he podido culminar esta obra y llegar a cumplir mi sueño de ser ingeniero. Por ello, muchísimas gracias. Me lo habéis dado todo en esta vida y ahora es tiempo de que os devuelva el favor.

A mi tutor Antonio, muchísimas gracias por confiar en mis capacidades para llevar a cabo este proyecto, por tu trabajo y tu paciencia a la hora de responder la gran cantidad de preguntas y problemas que me tenían perdido varias veces. A Paco, por su ayuda con el ecualizador gráfico que nos ha vuelto a todos locos y su amabilidad para ayudarme siempre. A mi tutor de la beca en ICS, Nicolás, por interesarse por mi proyecto y por su ayuda en la recta final. Gracias a ti he aprendido muchísimo en este último año de carrera.

Gracias a todos mis amigos, tanto los de dentro como los de fuera de la universidad, a los que siguen aquí ya los que ya se han ido. Gracias a vosotros he vivido miles de experiencias maravillosas, y espero que esas experiencias continúen en el futuro. Espero haber estado a la altura todos estos años de estudio, y espero dar la talla en el futuro que me depara. Gracias especiales a Iván, amigo mío desde que éramos pequeños. A Eduardo, Marcos y Elena por vuestro apoyo dentro y fuera de la universidad. A David y Yannick, mis dos recientes pero grandes amigos, nos queda mucho por vivir juntos. Por nuestra pasión por la música y el sonido, Christian, Diego, Dani y Abdón. Gracias especiales también a Ana, fiel compañera en mis inicios en la universidad y una de las mejores personas que conozco, y a Sara y Laura, dos de los grandes apoyos de mi vida, vuestra amistad no tiene valor. A los grandes músicos que forman mi grupo, espero que lleguemos lejos todos, porque jamás he visto en persona reunido tanto talento.

Finalmente, a mis tíos y mis queridos primos, gracias por todo vuestro apoyo. Estoy deseando veros crecer. A mi hermana, quiero desearte suerte y fuerza para la vida.

Y a todos los que no he nombrado y han estado aquí en mi vida, gracias por todo lo que me habéis dado y por todo lo que he aprendido con vosotros.

RESUMEN

La tecnología moderna de computación ha permitido cambiar radicalmente la investigación tecnológica en todos los ámbitos. El proceso general utilizado previamente consistía en el desarrollo de prototipos analógicos, creando múltiples versiones del mismo hasta llegar al resultado adecuado. Este es un proceso costoso a nivel económico y de carga de trabajo. Es por ello por lo que el proceso de investigación actual aprovecha las nuevas tecnologías para lograr el objetivo final mediante la simulación. Gracias al desarrollo de *software* para la simulación de distintas áreas se ha incrementado el ritmo de crecimiento de los avances tecnológicos y reducido el coste de los proyectos en investigación y desarrollo.

La simulación, por tanto, permite desarrollar previamente prototipos simulados con un coste mucho menor para así lograr un producto final, el cual será llevado a cabo en su ámbito correspondiente. Este proceso no sólo se aplica en el caso de productos con circuitería, si bien es utilizado también en productos programados. Muchos de los programas actuales trabajan con algoritmos concretos cuyo funcionamiento debe ser comprobado previamente, para después centrarse en la codificación del mismo. Es en este punto donde se encuentra el objetivo de este proyecto, simular algoritmos de procesado digital de la señal antes de la codificación del programa final.

Los sistemas de audio están basados en su totalidad en algoritmos de procesado de la señal, tanto analógicos como digitales, siendo estos últimos los que están sustituyendo al mundo analógico mediante los procesadores y los ordenadores. Estos algoritmos son la parte más compleja del sistema, y es la creación de nuevos algoritmos la base para lograr sistemas de audio novedosos y funcionales. Se debe destacar que los grupos de desarrollo de sistemas de audio presentan un amplio número de miembros con cometidos diferentes, separando las funciones de programadores e ingenieros de la señal de audio. Es por ello por lo que la simulación de estos algoritmos es fundamental a la hora de desarrollar nuevos y más potentes sistemas de audio.

Matlab es una de las herramientas fundamentales para la simulación por ordenador, la cual presenta utilidades para desarrollar proyectos en distintos ámbitos. Sin embargo, en creciente uso actualmente se encuentra el *software* Simulink, herramienta especializada en la simulación de alto nivel que simplifica la dificultad de la programación en Matlab y permite desarrollar modelos de forma más rápida. Simulink presenta una completa funcionalidad para el desarrollo de algoritmos de procesado digital de audio. Por ello, el objetivo de este proyecto es el estudio de las capacidades de Simulink para generar sistemas de audio funcionales. A su vez, este proyecto pretende profundizar en los métodos de procesado digital de la señal de audio, logrando al final un paquete de sistemas de audio compatible con los programas de edición de audio actuales.

ABSTRACT

Modern computer technology has dramatically changed the technological research in multiple areas. The overall process previously used consisted of the development of analog prototypes, creating multiple versions to reach the proper result. This is an expensive process in terms of an economically level and workload. For this reason actual investigation process take advantage of the new technologies to achieve the final objective through simulation. Thanks to the software development for simulation in different areas the growth rate of technological progress has been increased and the cost of research and development projects has been decreased.

Hence, simulation allows previously the development of simulated protoypes with a much lower cost to obtain a final product, which will be held in its respective field. This process is not only applied in the case of circuitry products, but is also used in programmed products. Many current programs work with specific algorithms whose performance should be tested beforehand, which allows focusing on the codification of the program. This is the main point of this project, to simulate digital signal processing algorithms before the codification of the final program.

Audio systems are entirely based on signal processing, both analog and digital systems, being the digital systems which are replacing the analog world thanks to the processors and computers. This algorithms are the most complex part of every system, and the creation of new algorithms is the most important step to achieve innovative and functional new audio systems. It should be noted that development groups of audio systems have a large number of members with different roles, separating them into programmers and audio signal engineers. For this reason, the simulation of this algorithms is essential when developing new and more powerful audio systems.

Matlab is one of the most important tools for computer simulation, which has utilities to develop projects in different areas. However, the use of the Simulink software is constantly growing. It is a simulation tool specialized in high-level simulations which simplifies the difficulty of programming in Matlab and allows the developing of models faster. Simulink presents a full functionality for the development of algorithms for digital audio processing. Therefore, the objective of this project is to study the posibilities of Simulink to generate funcional audio systems. In turn, this projects aims to get deeper into the methods of digital audio signal processing, making at the end a software package of audio systems compatible with the current audio editing software.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

Anexo A

Anexo B

ÍNDICE DE TABLAS

GLOSARIO DE ACRÓNIMOS

#	
3D	Three Dimensions

A	
A/D	Analog/Digital
APG	Audio Plugin Generator
AT	Attack Time

B	
BL	Blend
BPM	Beat per minute

D	
D/A	Digital/Analog
DAW	Digital Audio Workstation
DFT	Discrete Fourier Transform
DJ	Disc Jockey
DLL	Dynamic Link Library
DSP	Digital Signal Processor

F	
FB	Feedback
FIR	Finite Impulse Response
FF	Feedforward
FFT	Fast Fourier Transform
IFFT	Inverse Fast Fourier Transform
FX	Special Effects

G	
GUI	Graphical User Interface

I	
IIR	Infinite Impulse Response
ISO	International Standards Organization
ISR	Interrupt Service Routine

L	
LTI	Linear Time Invariant

O	
OLA	Overlap and Add

P	
PC	Personal Computer

R	
RT	Release Time

S	
SDK	Software Development Kit
SOLA	Synchronous Overlap and Add
STFT	Short-Time Fourier Transform

U	
UIC	User Interface Configuration

V	
VST	Virtual Studio Technology

W	
WAV	Waveform Audio Filter Format
WAVE	Waveform Audio Filter Format

0. OBJETIVOS

- Aprendizaje y manejo de la herramienta Simulink, como *software* de programación junto a Matlab y como herramienta de procesado de señales.

- Construcción de diferentes algoritmos de procesado digital de audio mediante Simulink, orientados a su uso en formato *plugin*.

- Comprobar las capacidades y limitaciones de Simulink para la creación de prototipos de *plugins* de audio.

- Comprobar las capacidades y limitaciones de la herramienta *APG* como nexo de unión entre el código en Simulink y el código utilizado por *plugins*.

- Profundizar en los algoritmos actuales de los sistemas de procesado digital de audio. Las áreas a cubrir son el procesado en frecuencia, procesado de dinámica, procesado de retardos y compresión-expansión temporal.

- Profundizar en el procesado digital de la señal de audio y las diferentes técnicas aplicables al diseño de algoritmos.

- Desarrollar un paquete de *software* de audio funcional formado por los diferentes *plugins* construidos en Simulink y transformados mediante la herramienta *APG*. Este paquete debe ser compatible con las herramientas modernas de edición y mezcla de audio.

- Probar el funcionamiento y la utilidad de los algoritmos desarrollados como *plugins* con señales musicales y de voz, buscando el correcto funcionamiento de los mismos y la ausencia de artificios.

1. INTRODUCCIÓN AL PROCESADO DIGITAL DE LA SEÑAL

1.1. INTRODUCCIÓN

El procesado digital de la señal es la herramienta fundamental actualmente para trabajar con la señal de audio. En la cadena de la señal de audio, tanto en su origen como a la hora de ser reproducida, se trata de una señal de naturaleza analógica. Las señales analógicas son continuas, presentan un valor para cada instante de tiempo, por lo que resulta imposible trabajar con ellas en un procesado que no sea en tiempo real ya que no pueden ser almacenadas. Además, el procesado digital permite conseguir procesos y algoritmos mucho más complejos y funcionales que el procesado analógico. Gracias a la creciente investigación y mejora de los componentes semiconductores y los ordenadores, estas capacidades siguen incrementándose año tras año, obteniéndose una mayor velocidad y complejidad de procesado a un coste mucho menor.

Trabajar con procesados digitales implica obtener señales digitales a partir de las analógicas, o generar directamente señales digitales. Por ello es necesario muestrear y cuantizar las señales analógicas mediante convertidores analógico/digitales, tomando valores a una **frecuencia de muestreo** determinada. Si la señal fuese generada por ordenador, sería también necesario generar valores a intervalos de tiempo equiespaciados según el periodo de muestreo. El **teorema de muestreo de Nyquist** establece la limitación por la cual una señal está unívocamente representada si la frecuencia de muestreo es dos veces la frecuencia máxima de la señal.

$$f_s > 2f_M \qquad con\ f_s = \frac{1}{T_s} \tag{1}$$

A la salida de un sistema de procesado, la señal debe convertirse de nuevo a analógica mediante convertidores digital/analógicos. De esta forma, queda completado el sistema genérico de procesado digital, tal y como muestra la figura [1]. Esta estructura puede variar dependiendo de la aplicación. Si la señal de entrada ya hubiese sido almacenada digitalmente o si se desease almacenar la señal en lugar de ser reproducida, no serían necesarios los convertidores a la entrada y a la salida.

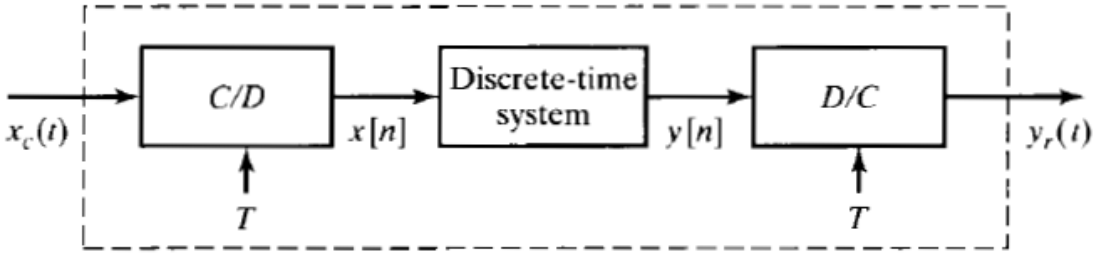

Figura 1: Estructura general de un sistema de procesado digital de la señal.

1.2. TRANSFORMADA DISCRETA DE FOURIER (DFT)

La transformada de Fourier es el cálculo capaz de dar la información de una señal en el dominio de la frecuencia, dando lugar a una función continua periódica de período 2π. Esta función continua puede ser representada y almacenada de forma discreta mediante la **transformada discreta de Fourier** o **DFT**, cuyo resultado es una secuencia de muestras correspondientes a la transformada de Fourier de una señal. Las expresiones para el cálculo de la DFT y la DFT inversa se obtienen a partir de la serie discreta de Fourier [ref. 6], y se corresponde con las expresiones [2] y [3], siendo N el periodo de la DFT, el cual se corresponde con el número de puntos tomados para el cálculo.

$$X[k] = \frac{1}{N}\sum_{k=0}^{N-1} x[n]e^{-jk\frac{2\pi}{N}n} \qquad 0 \leq k \leq N-1 \tag{2}$$

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[K]e^{jk\frac{2\pi}{N}n} \qquad 0 \leq k \leq N-1 \tag{3}$$

Generalmente, en los algoritmos de procesado digital se utiliza un algoritmo más eficiente conocido como la **transformada rápida de Fourier** o **FFT**. Este algoritmo reduce considerablemente el número de operaciones a realizar con respecto a la DFT gracias a una serie de transformadas de menor longitud agrupadas entre sí. Los algoritmos FFT suelen calcularse con señales de longitud potencias de 2 (1024, 2048, 4096…), por lo que a señales de longitudes menores deben añadirse ceros al final de la secuencia.

1.3. TRANSFORMADA Z

La transformada Z es el equivalente en el dominio discreto de la transformada de Laplace en el dominio continuo. Ambas transformadas están estrechamente ligadas a la transformada de Fourier, y permiten obtener la **función de transferencia** H(s) o H(z) de un sistema caracterizado por su respuesta al impulso. La transformada Z se calcula mediante la expresión [5], muy similar a la expresión de la transformada de Fourier [4].

$$X\left(e^{j\omega}\right) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} \tag{4}$$

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n} \tag{5}$$

La variable z es una variable compleja y continua estrechamente ligada a la transformada de Fourier. Realmente, si $z = e^{j\omega}$ la transformada Z es equivalente a la transformada de Fourier. La utilidad de utilizar esta transformada es la obtención de la función de transferencia H(z) de un sistema, definida como la transformada Z de la señal de salida dividida por la transformada Z de la señal de entrada, expresión [6]. La transformada Z puede visualizarse mediante el **plano Z**, que se corresponde con la circunferencia unidad donde se representan los **polos** y **ceros** del sistema. El ángulo de un punto de este diagrama con respecto al eje real se corresponde con la pulsación discreta ω (rad).

$$H(z) = \frac{Y(z)}{X(z)} \tag{6}$$

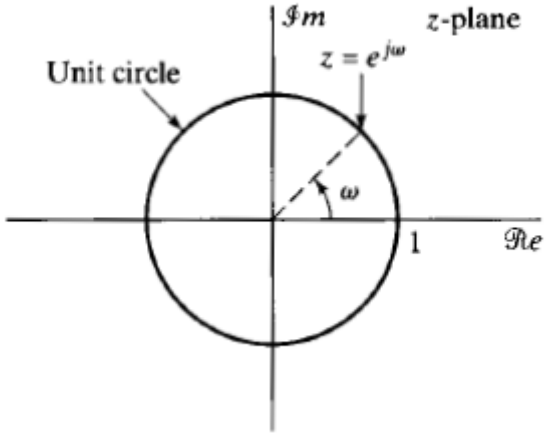

Figura 2: Diagrama de polos y ceros en el plano Z.

1.4. FILTROS DIGITALES

Los filtros digitales son una de las herramientas más utilizadas en el procesado digital, siendo fundamentales para un gran abanico de aplicaciones de diversos ámbitos. Existen dos clases de filtros digitales, filtros **IIR** (*Infinite Impulse Response*) y filtros **FIR** (*Finite Impulse Response*), cada uno de ellos con propiedades diferentes. Los filtros digitales son sistemas **LTI** (lineales e invariantes en el tiempo) caracterizados, por tanto, por su respuesta al impulso h[n]. Generalmente son utilizados por sus características en el dominio de la frecuencia, por lo que se suele prestar especial atención a su transformada en frecuencia $H(e^{j\omega})$. Los filtros digitales pueden ser representados de forma gráfica mediante las estructuras en diagramas de bloques, las cuales indican la disposición de las señales a emplear, los registros de almacenamiento, las ganancias y las unidades de retardo para construir la función de transferencia a partir de la ecuación en diferencias, cuya expresión general se corresponde con la ecuación [7].

$$y[n] = \sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{M} b_k x[n-k] \quad (7)$$

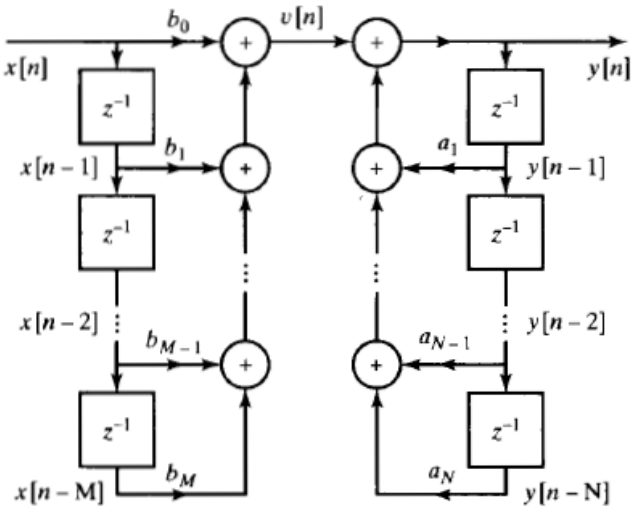

Figura 3: Estructura de cálculo de un filtro digital (forma directa I).

Según el tipo de filtro deseado, el proceso de diseño de filtros digitales es muy diferente. Para el diseño de filtros IIR se utiliza principalmente la **transformada bilineal**. Esta transformada parte de las especificaciones del filtro en el dominio discreto ω (rad), aplicando la transformada bilineal de las especificaciones para obtener las especificaciones en el dominio analógico Ω (rad/s). A partir de estas especificaciones se caracteriza el filtro mediante su función de transferencia analógica H(s), y se aplica la transformada bilineal para obtener la función de transferencia digital del filtro H(z) [ref. 6].

Los filtros de respuesta finita o FIR presentan una característica muy importante y útil en el mundo del audio. Se trata de filtros capaces de conseguir respuestas de fase lineales en la banda de paso, evitando así la distorsión de fase que genera efectos audibles en la señal tratada. Son diseñados mediante el método de la **transformada de Fourier** o **de la ventana**. Este método consiste en enventanar la respuesta al impulso de las especificaciones ideales del filtro multiplicando por una ventana seleccionada previamente (h[n]•w[n]), realizando a posteriori la transformada discreta de Fourier para obtener la respuesta en frecuencia del filtro digital. Existen distintos tipos de ventanas, cada una de las cuales consigue unos resultados diferentes. Deben ser seleccionadas en función de las características del filtro que se desea conseguir.

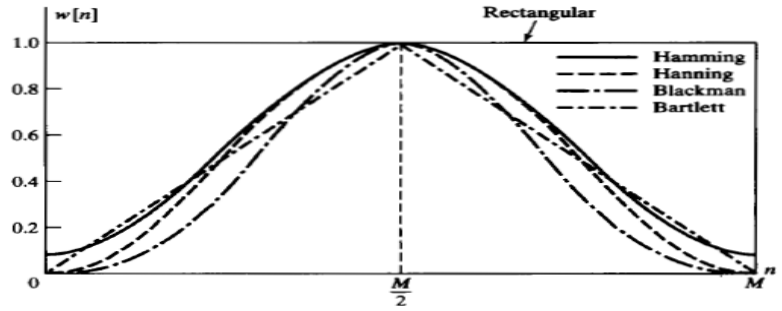

Figura 4: Respuesta al impulso de diferentes ventanas utilizadas en el proceso de diseño de filtros FIR.

1.5. INTERPOLACIÓN Y DIEZMADO

Los procesos de interpolación y diezmado son muy comunes en el procesado digital cuando se pretende realizar cambios en la frecuencia de muestreo de la señal, aunque son utilizados también en muchas otras aplicaciones. Ambos procesos producen efectos diferentes en el espectro de la señal y deben ser estudiados para corregirlos.

El proceso de **diezmado** consiste en eliminar una de cada N muestras de una secuencia de entrada. Este proceso reduce la frecuencia de muestreo de la señal en un factor 1/N. En el espectro, cada réplica del espectro periódico se ensancha N veces, produciendo solape entre las réplicas si el espectro cubre una banda mayor a π/N (rad). Para evitar los solapes espectrales se añade un filtro paso bajo antes de realizar el proceso de diezmado.

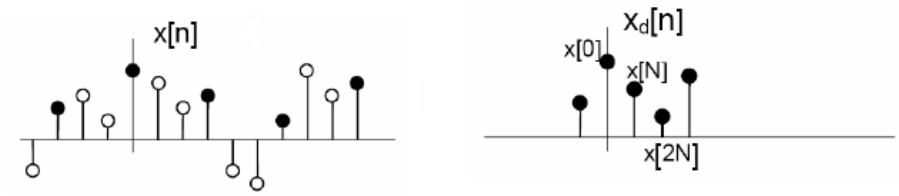

Figura 5: Secuencia de entrada antes y después del proceso de diezmado.

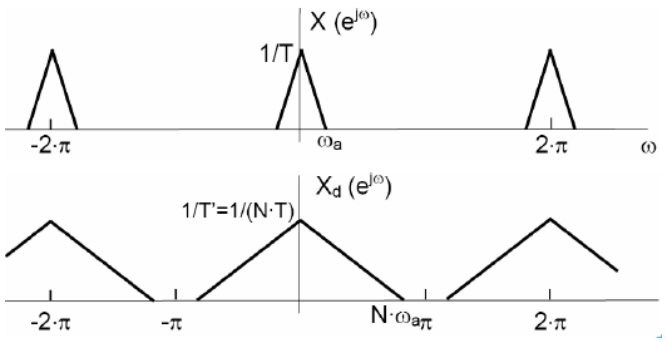

Figura 6: Respuesta en frecuencia ensanchada tras el proceso de diezmado.

Por el contrario, el proceso de **interpolación** incrementa la frecuencia de muestreo de una secuencia en un factor L mediante la inserción de L-1 ceros entre cada muestra, realizando a continuación la interpolación entre muestras mediante un filtro paso bajo. La inserción de ceros provoca que las réplicas espectrales se estrechen, pasando a estar presentes un número mayor de réplicas. El filtro paso bajo interpola, eliminando las réplicas espectrales tal y como muestra la figura [8].

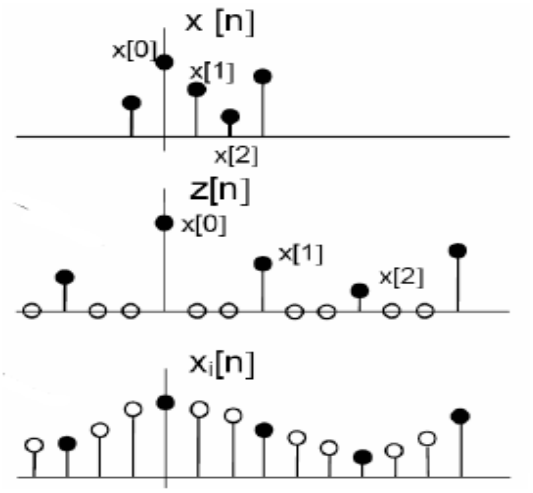

Figura 7: Secuencia de entrada tras el proceso de inserción de ceros e interpolación.

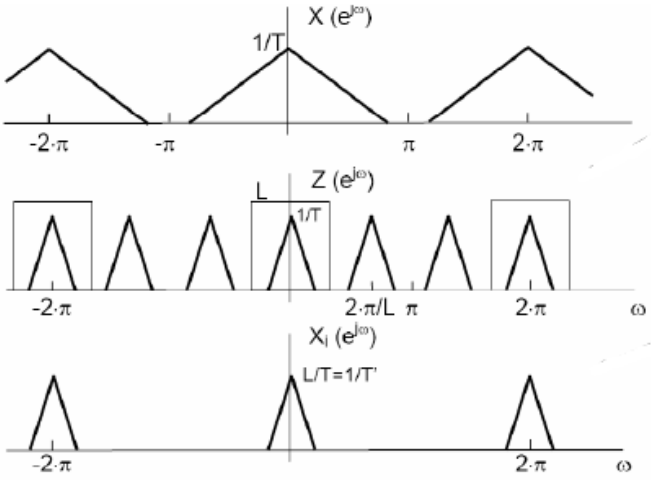

Figura 8: Réplicas producidas por el proceso de inserción de ceros y proceso de interpolación en el espectro mediante filtrado paso bajo.

En ambos casos, la frecuencia de corte del filtro debe ser π/N, π/L. En el caso de desear un cambio en la frecuencia de muestreo por un factor no entero, deben aplicarse ambos métodos. El orden es importante, debiendo realizarse en primer lugar la inserción de ceros, seguido del filtro paso bajo más restrictivo entre ambos y del proceso de diezmado. Además el filtro debe tener una ganancia en la banda de paso igual al factor de interpolación L, ya que el proceso de inserción de ceros atenúa el nivel de la señal.

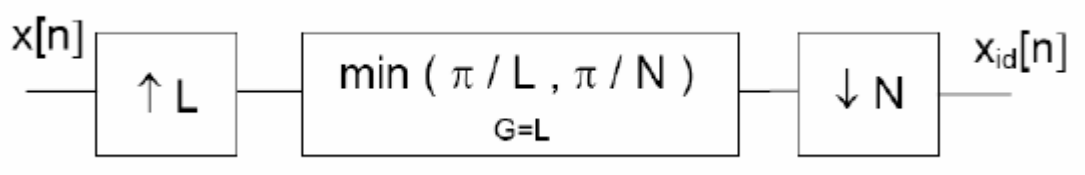

Figura 9: Proceso de remuestreo de una señal por un factor no entero.

1.6. CONCLUSIONES

Este capítulo sirve como una introducción muy resumida a las técnicas de procesado digital utilizadas por este proyecto. Existen muchos más detalles acerca de los procesos de filtrado, transformadas y herramientas de procesado en las referencias [ref. 6] y [ref. 8]. El lector puede profundizar más en estos aspectos de la teoría de señales en dominios continuo y discreto.

A su vez, este proyecto profundiza en estos conocimientos a la hora de ser utilizados en las aplicaciones de audio digital y en los algoritmos desarrollados, siendo la teoría de la señal y el procesado digital bases fundamentales de estos algoritmos y fuentes de nuevos algoritmos que siguen surgiendo en la actualidad y seguirán surgiendo en el futuro.

El estado del arte del procesado digital es un ámbito en constante crecimiento, gracias a las nuevas tecnologías en computación y DSPs y a las nuevas aplicaciones en las que están siendo aplicados. Cada día cubren un abanico más amplio como son el reconocimiento de patrones, la biomedicina, el procesado de imágenes y voz, telecomunicaciones, señales de radares, etc… sustituyendo los procesos analógicos más costosos y difíciles de implementar mediante circuitería. El procesado digital de la señal es ya el presente y el futuro de las tecnologías, que combinado con la capacidad de los ordenadores y la programación es capaz de lograr prácticamente cualquier objetivo.

2. INTRODUCCIÓN A LA SIMULACIÓN MEDIANTE SIMULINK

2.1. INTRODUCCIÓN

Simulink es una herramienta de simulación presente dentro del software de programación Matlab, con una particularidad que la convierte en una herramienta muy potente. Mientras que Matlab presenta un sistema de programación mediante su propio código, Simulink ha enmascarado este código mediante una **programación visual** de un nivel superior. La programación visual, similar a software de desarrollo como *LabView*, permite concatenar distintas funciones creando un flujo de datos en el programa mediante bloques, de forma que se construye un **diagrama de bloques funcional**.

En su interior, cada uno de los bloques que conforman las librerías de Simulink son funciones desarrolladas en Matlab, por lo que permite perfectamente el intercambio de datos entre Matlab y Simulink, pudiendo realizar tratamientos y análisis del sistema en ambos entornos. Existen multitud de librerías para distintas aplicaciones, las cuales abarcan desde sistemas de comunicación y procesado de la señal hasta simulación en 3D, lógica, redes neuronales, etc. La posibilidad de poder crear y definir librerías propias aumenta aún más el rango de capacidad de Simulink y lo convierte en una herramienta potente y fundamental, tanto a nivel educativo como a nivel de producción en el ámbito tecnológico.

Al igual que *Labview*, Simulink permite la interacción con elementos de *hardware* externos al PC, como pueden ser placas de procesado DSP, Arduino o Lego. Los modelos construidos en Simulink pueden ser ejecutados en placas compatibles, o interactuar con ellos mediante sistemas de entrada/salida.

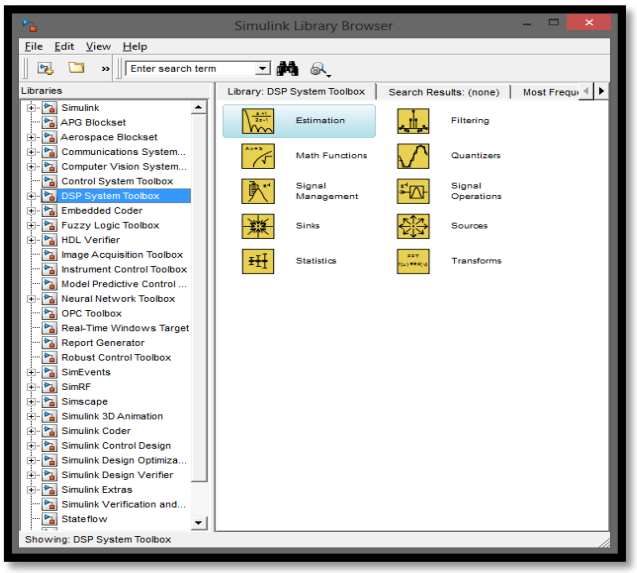

Figura 10: Librerías de bloques en Simulink.

Simulink presenta, por tanto, una serie de ventajas por las que es ampliamente utilizado. Sin embargo, no todo es perfecto, por lo que existen desventajas las cuales deben tenerse en cuenta a la hora de programar diagramas de bloques.

Las principales ventajas son:

- Simulación en tiempo real mediante flujo de datos.
- Interfaz de programación rápida e intuitiva.
- Entorno de programación por bloques, forma de programar cómoda y habitual para ingenieros.
- Programación de forma rápida de procesos complicados, evitando las complicaciones intrínsecas en la programación mediante código como son las definiciones, funciones, bucles, etc…
- Interacción entre Simulink y Matlab.
- Compatibilidad con *hardware* externo (DSPs, Arduino y Lego).
- Posibilidad de definición de librerías y bloques propios mediante código Matlab y C.
- Visualización de resultados en tiempo real dentro del modelo.
- Facilidad y rapidez de interpretación de un modelo frente a un archivo de código debido a la visualización gráfica del diagrama de bloques.

Las dos principales desventajas son:

- Código de ejecución más lento en la máquina que otros códigos como C/C++.
- Dependiendo de la aplicación, programación de un nivel superior al necesario, por lo que se recurre frecuentemente a la programación en Matlab.

2.2. BASES DE LA PROGRAMACIÓN EN SIMULINK

En [ref. 16] la página oficial de Matlab/Simulink presenta distintos tutoriales para adaptarse a las bases de la programación de modelos en Simulink. Sin embargo, para la facilidad de lectura de este libro se dan algunas pautas básicas acerca de este tipo de programación, centrándose en el uso de la librería de procesado digital de señal *DSP System Toolbox*. La instalación de Simulink incluye una amplia biblioteca de ayuda accesible rápidamente mediante el entorno de programación. Se recomienda consultar cada bloque a utilizar para conocer su funcionamiento en profundidad.

En primer lugar, es importante realizar la configuración del modelo, accesible desde el panel superior mediante la opción *Model Configuration Parameters*. Debe configurarse el modelo según se trate de un sistema con escalones variables (*Variable-Step)* o fijos (*Fixed-Step)*. Simulink trabaja con un eje temporal el cual ejecuta el modelo en intervalos de un escalón determinado, el cual puede ser fijo o variable. En el caso del desarrollo de algoritmos de audio los cuales van a ser compilados en código C, es

necesario un sistema de escalón fijo ya que se trabaja a una frecuencia de muestreo fija. Debe seleccionarse también esta frecuencia de muestreo. La opción *auto* delega en Simulink la selección de la frecuencia en función de las señales del modelo. Finalmente, en función de la simulación a realizar debe seleccionarse si se trata de un sistema continuo o discreto. Los sistemas continuos se basan en realizar la integración temporal para el cálculo en todos los intervalos de tiempo. Para el caso de los algoritmos de audio, estos son digitales por lo que el sistema deberá ser configurado como discreto.

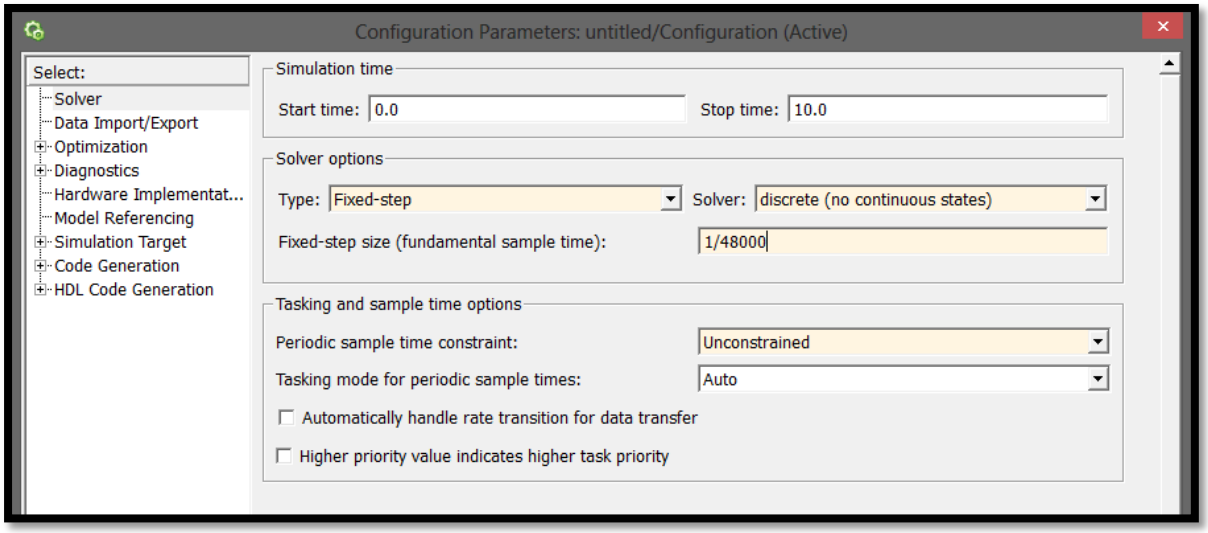

Figura 11: Ventana de configuración del modelo en Simulink.

Por convenio, generalmente las estructuras sencillas se interpretan de izquierda a derecha, estando presente las señales de entrada a la izquierda y las posibles señales de salida a la derecha. Es necesario definir estas señales, ya sean constantes, archivos de audio, etc… En los subapartados de las librerías *Sources* pueden encontrarse los bloques que actúan como fuentes en el modelo. Simulink, al igual que Matlab, trabaja mediante matrices. En Simulink, cada columna representa un **canal de datos** (en el caso de audio cada columna se corresponde con los canales de audio). Los bloques de Simulink están configurados para trabajar con cada canal por separado.

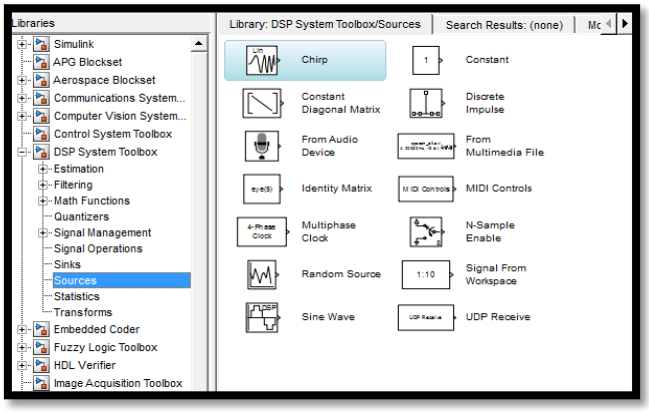

Figura 12: Bloques fuente de señal en Simulink.

Existen dos clases de flujo de datos en Simulink, las cuales deben tenerse en cuenta a la hora de realizar el modelo. Los bloques pueden configurarse para trabajar muestra a muestra o con conjuntos de muestras o ***frames***. En el primer caso, el flujo del modelo se denomina ***Sample-based***, mientras que en el segundo caso el flujo se denomina ***Frame-based***. Los modelos mediante *frames* son utilizados en sistemas de tiempo real, y su principal ventaja es mejorar el rendimiento y acelerar el procesado del modelo. Esto es debido al proceso de interrupción al adquirir datos (*ISR, Interrupt Service Routine)*. En el caso de trabajar muestra a muestra, el sistema de adquisición de datos se interrumpe por cada muestra. La muestra es procesada por el sistema y entregada a la salida tras un tiempo determinado en función del modelo. Al utilizar *frames*, el sistema de adquisición de datos se interrumpe en un número de veces inferior, acelerando el sistema. En este caso debe tenerse en cuenta el tiempo de latencia, ya que seleccionar *frames* de un gran número de muestras puede provocar una latencia demasiado elevada.

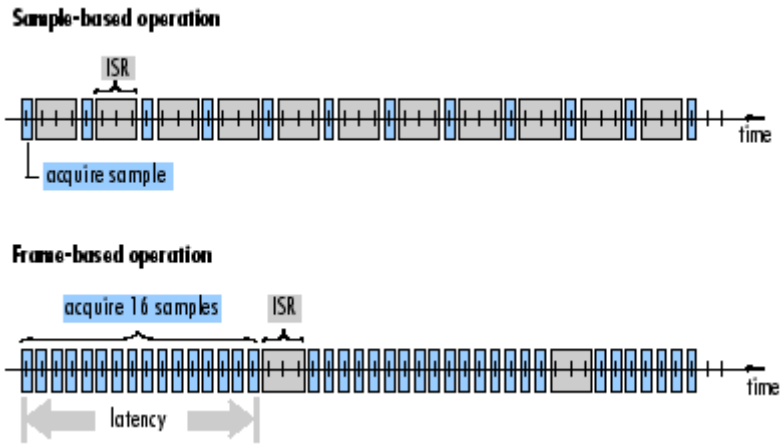

Figura 13: Sample-based Vs Frame-based.

2.3. FUNCIONALIDADES DE SIMULINK

Simulink no sólo presenta sus librerías específicas como funcionalidades. Sus posibilidades incluyen la creación de sistemas lógicos mediante bloques activables, así como la definición de funciones y boques propios. El subapartado de la librería general de Simulink *Ports & Subsystems* permiten establecer distintos niveles dentro del modelo mediante **subsistemas**. Estos subsistemas, a su vez, pueden estar activados mediante una determinada condición, entre las que se encuentran las opciones *if, switch, trigger* o *enable.* Un subsistema con cualquiera de estos bloques será activado si la señal seleccionada cumple una determinada condición o sea activado (*triggered* o *enable*) mediante una señal de control.

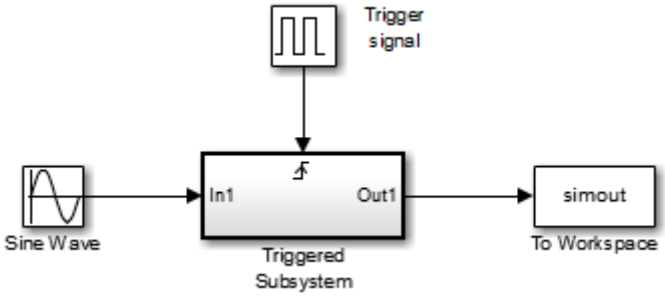

Figura 14: Subsistema activable mediante una señal de control.

A su vez, Simulink permite crear **máscaras** en los subsistemas, pudiendo definir bloques propios. Las máscaras establecen una serie de variables que actúan como globales dentro del subsistema, y pueden configurarse desde fuera accediendo a la configuración del bloque. Una vez se selecciona el nombre de una variable, será accesible desde cualquier lugar del subsistema haciendo mención a su nombre, incluso dentro de otros subsistemas que conforman niveles inferiores. Este proceso resulta muy cómodo para programar modelos con una estructura complicada, siendo equivalente a la subdivisión en funciones utilizada en programación, que divide el algoritmo en procesos más sencillos.

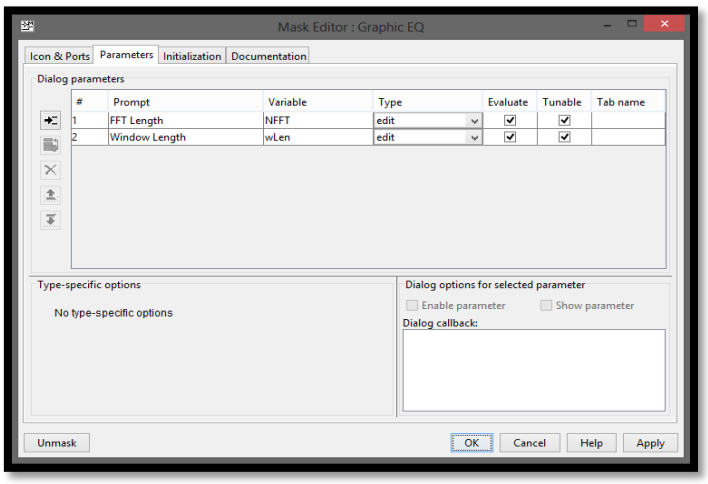

Figura 15: Ventana de configuración de máscaras en Simulink.

Mientras que Matlab o cualquier otro *software* de programación presentan un sistema de control de errores o *Debugger*, su implementación en Simulink es más complicada, ya que se trata de un sistema que en muchas ocasiones trabaja en tiempo real. Es necesario para comprobar el correcto funcionamiento del sistema, utilizar los visualizadores de señal presentes en los subapartados *Sink* de las librerías. Existen *displays* de valores, osciloscopios en diferentes dominios (tiempo/frecuencia), e incluso la posibilidad de reproducir una señal mediante altavoces. Por otro lado, en el caso de errores Simulink informa del bloque donde se está produciendo un error y da información del error causado, con ayudas muy útiles como mostrar en todos los puntos los tamaños

de las matrices de datos que se están propagando por el sistema. Existe también la posibilidad de utilizar la simulación **paso a paso** si se desea comprobar el funcionamiento del sistema en un instante de tiempo determinado.

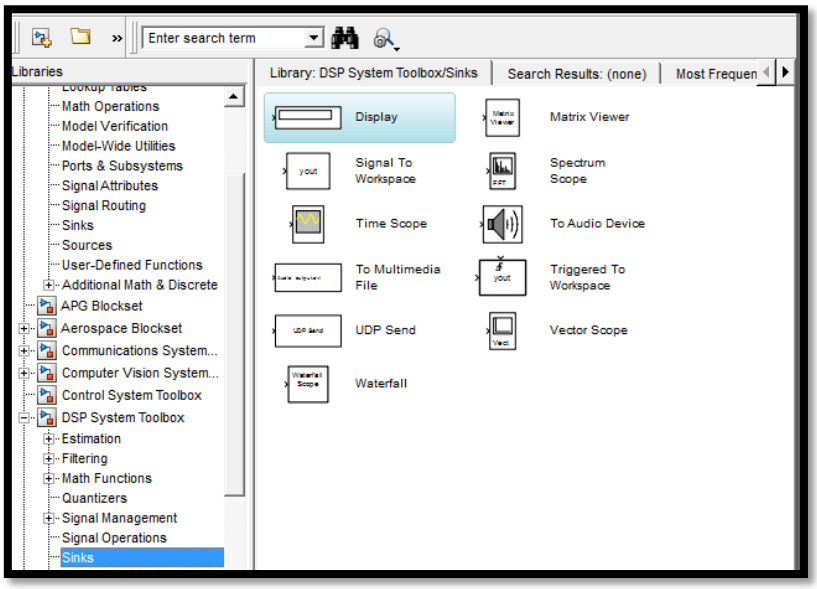

Figura 16: Bloques visualizadores de señal en Simulink.

2.4. CONCLUSIONES

Las características descritas anteriormente sirven como introducción a los algoritmos desarrollados en este proyecto. En cada uno de los modelos que implementan estos algoritmos se profundiza más en el uso concreto de bloques y características específicas del procesado digital en Simulink, así como de bloques de uso común en otros ámbitos. Además, este proyecto hace especial hincapié en destacar para cada algoritmo las posibilidades de Simulink para desarrollar algoritmos de procesado digital de audio, y los inconvenientes y complicaciones que han surgido durante el desarrollo de sendos algoritmos.

Por otro lado, Simulink se presenta como una herramienta aparentemente ideal para el cometido del desarrollo de algoritmos de audio, tanto por sus capacidades intrínsecas como por su mayor sencillez de programación para ingenieros, acostumbrados a trabajar con diagramas de bloques.

3. PROCESADO EN FRECUENCIA

3.1. INTRODUCCIÓN

Uno de los tratamientos del audio más importantes a la hora de procesar la señal es el procesado en frecuencia. El control adecuado de la señal en el dominio de la frecuencia es fundamental para obtener el sonido deseado en función de la aplicación en la que va a ser utilizado. Así, los ecualizadores permiten realizar este control de la señal mediante la implementación de distintos tipos de filtros. Los tipos de filtros más comunes son:

- **Filtros paso bajo, paso alto y paso banda:** Como su propio nombre indica, se trata de filtros que permiten el paso de un margen de frecuencias determinado por la frecuencia de corte, atenuando las frecuencias que se encuentran fuera de este margen con una caída que depende del orden del filtro construido. Estos filtros se suelen incluir al principio o al final de un procesador de audio para eliminar efectos y artificios no deseados, o para seleccionar la banda de frecuencias a la que se desea aplicar un determinado procesado.

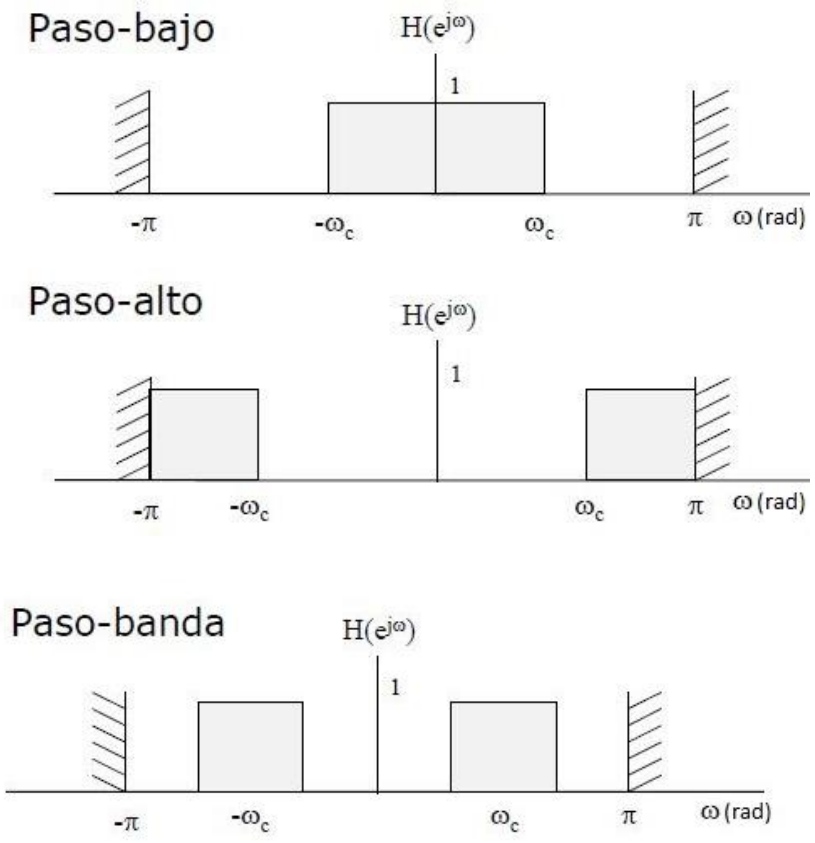

Figura 17: Filtros paso bajo, paso alto y paso banda en el dominio de la frecuencia discreta.

- **Filtros *Notch* o filtros ranura:** Son filtros de banda eliminada que permiten eliminar de manera selectiva un margen reducido de frecuencias. Requieren atenuaciones lo más grande posibles y anchos de banda muy reducidos. Son utilizados cuando se pretende eliminar ruidos de frecuencias muy concretas o para eliminar la realimentación acústica en un sistema de refuerzo sonoro.

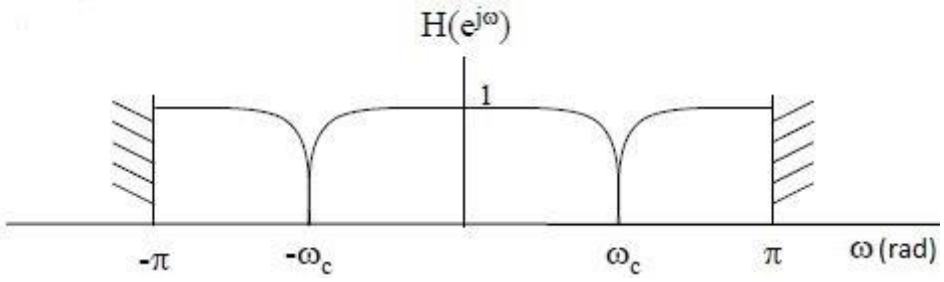

Figura 18: Filtro Notch o ranura en el dominio de la frecuencia discreta.

- **Control de tono o ecualizador tipo *Shelving*:** Son ecualizadores capaces de realzar o atenuar un margen amplio de frecuencias seleccionable, tanto en bajas como en altas frecuencias. De esta forma quedan las frecuencias medias sin tratar. Suelen presentar un realce o atenuación de ±12 dB y ganancia unidad fuera de la banda de paso.

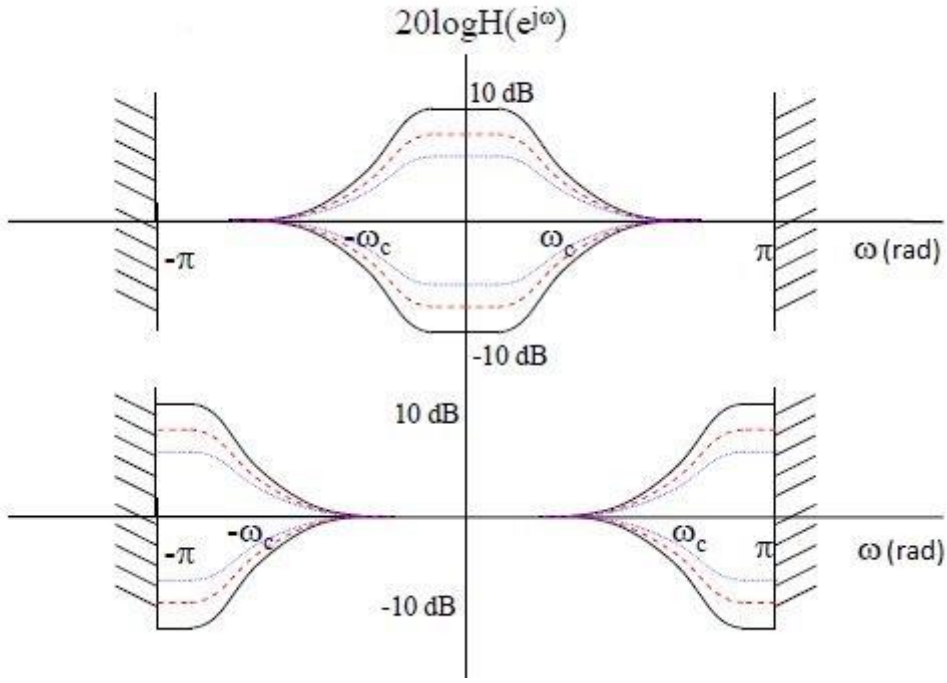

Figura 19: Ecualizadores tipo Shelving en el dominio de la frecuencia discreta. Low Shelving (arriba) y High Shelving (abajo).

- **Ecualizadores resonantes o ecualizador tipo *Peak*:** Son filtros paso banda con ganancia unidad fuera de la banda de paso, que permiten realzar o atenuar un margen de frecuencias alrededor de la frecuencia central o frecuencia de resonancia. La ganancia suele encontrarse en un margen de ±12 dB.

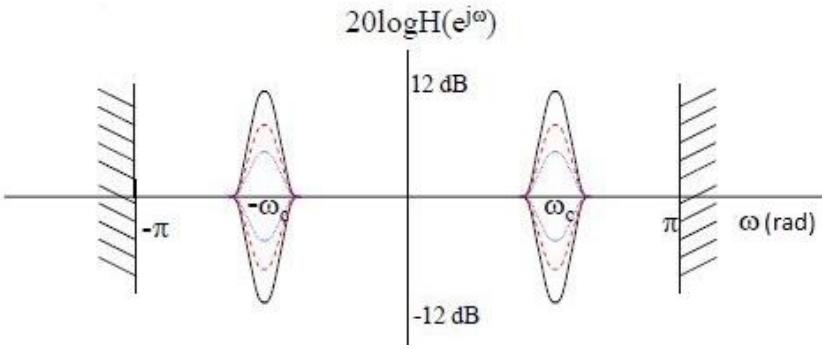

Figura 20: Filtros Peak en el dominio de la frecuencia discreta.

La configuración de los filtros que conforman un ecualizador se realiza mediante tres parámetros distintos. Dependiendo del tipo de ecualizador y su construcción, estos parámetros serán fijos o variables. En primer lugar, la **frecuencia de corte** permite seleccionar la sintonía del filtro, es decir, la frecuencia sobre la que se desea posicionar. Así, para los filtros paso bajo, paso alto y los ecualizadores *shelving*, la frecuencia de corte será aquella a la que comienza la pendiente del filtro. Existen distintas definiciones para la frecuencia de corte, siendo la más común aquella frecuencia a la que el nivel se ha reducido - 3dB con respecto al nivel en la banda de paso. Para filtros paso banda, filtros ranura y ecualizadores *peak*, la frecuencia de resonancia es la frecuencia central a la que actúan estos filtros. En segundo lugar, la **ganancia** permite controlar el realce o atenuación que presenta un filtro en la banda de paso. Por comodidad, la ganancia se establece en dB, obtenida a partir de la ganancia en escala lineal del filtro. Finalmente, el **factor de calidad** permite seleccionar el ancho de banda de un filtro, es decir, el margen de frecuencias sobre el que actúa. De nuevo existen distintas definiciones para el ancho de banda de un filtro, siendo la más común la banda de frecuencias donde la atenuación es igual o menor a -3dB.

Una vez establecidos los parámetros de configuración y los tipos de filtros, pueden definirse los efectos de audio utilizados para el procesado en frecuencia de una señal. Fundamentalmente existen tres tipos de ecualizadores distintos, definidos a continuación:

- **Ecualizadores gráficos:** Son un conjunto de filtros paso banda dispuestos en paralelo, cada uno de los cuales está centrado en una banda determinada según el número de filtros. Los más comunes son de 10, 20 y 30 bandas. Este ecualizador se caracteriza por no disponer de un control para sintonizar los filtros, es decir, no se puede cambiar la frecuencia a la que actúan. El único parámetro variable por el usuario suele ser el control de ganancia de cada uno de ellos. La principal utilidad de los ecualizadores gráficos es el control de la respuesta en frecuencia en una sala, ajustando el ecualizador para que la respuesta del sistema de refuerzo sonoro junto con la sala sea lo más plana posible.

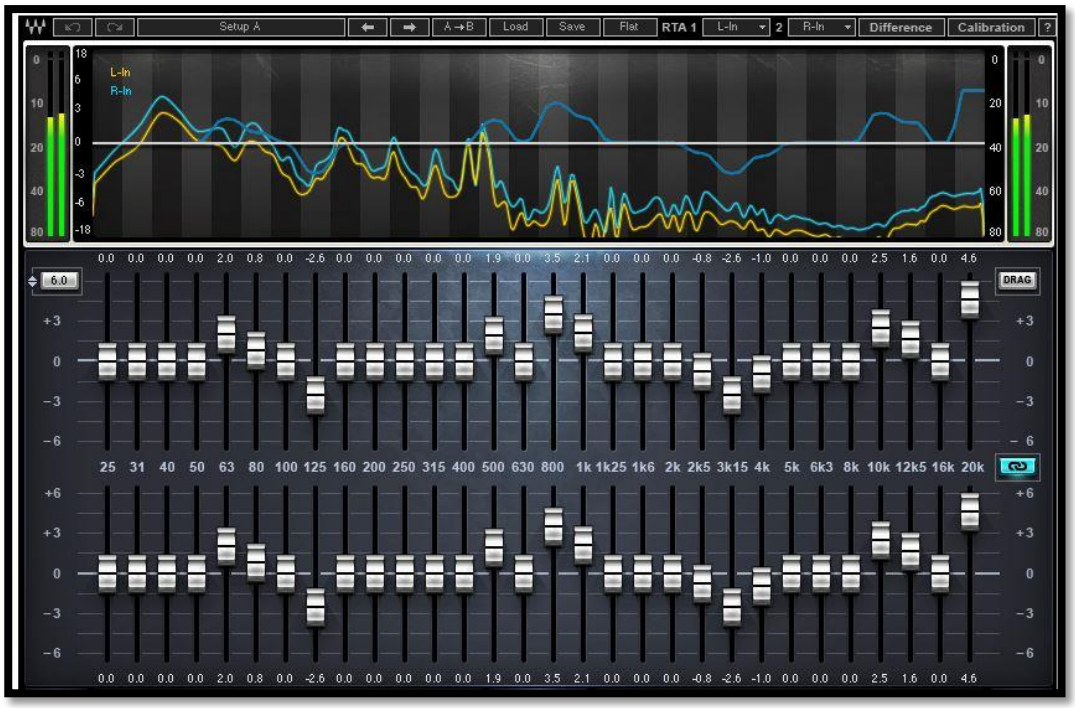

Figura 21: Ecualizador gráfico de la marca WAVES.

- **Ecualizadores paramétricos:** Se trata de un conjunto de ecualizadores dispuestos en cascada que permiten realizar un ajuste totalmente personalizado de la señal en frecuencia. Suelen estar formados por dos etapas de control de tonos o ecualizadores *shelving* y por una serie de filtros resonantes o ecualizadores *peak*. Los controles de estos ecualizadores permiten seleccionar la frecuencia, el factor de calidad y la ganancia de cada uno de los filtros.

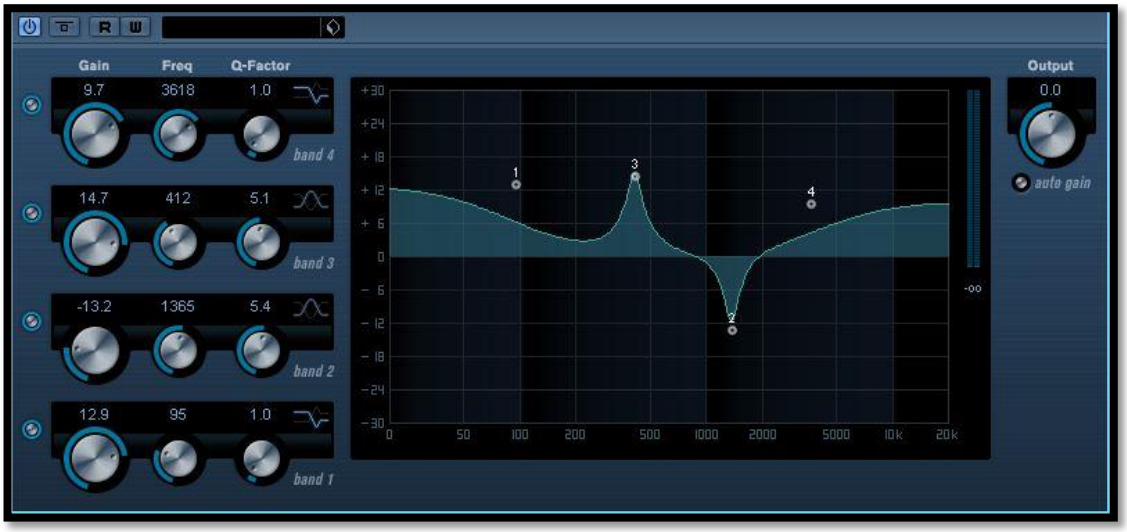

Figura 22: Plugin de un ecualizador paramétrico en Steinberg Cubase.

- **Ecualizadores semiparamétricos:** Similares a los ecualizadores paramétricos, presentan una estructura idéntica con una diferencia fundamental. Para permitir un ajuste más rápido y sencillo se elimina el control del factor de calidad. Este tipo de ecualizadores es muy utilizado sobre todo en los canales de entrada de una mesa de mezclas.

3.2. ECUALIZADOR PARAMÉTRICO

El diseño a implementar en Simulink consiste en un ecualizador paramétrico de 5 bandas, dos de las cuales se corresponden con controles de tono *Low Shelving* y *High Shelving*. Las tres bandas restantes se corresponden con filtros resonantes *peak*. Todos los filtros a implementar son filtros de **2º orden**. Se debe incluir también un estimador de la función de transferencia que permita visualizar la respuesta en frecuencia del sistema en tiempo real.

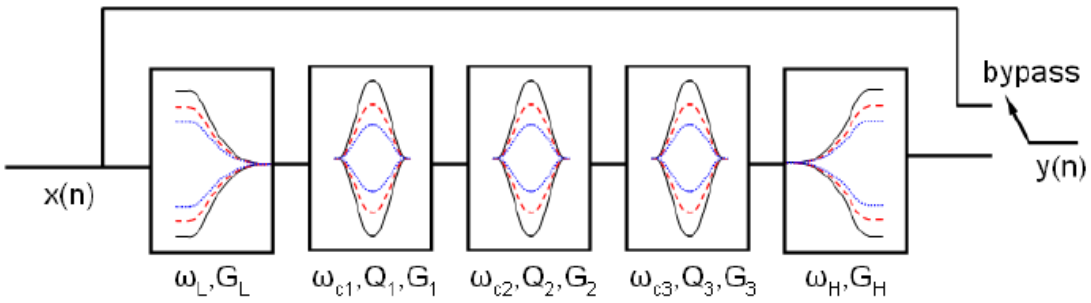

Figura 23: Diagrama de bloques de un ecualizador paramétrico.

3.2.1. DISEÑO DE ECUALIZADORES PEAK Y SHELVING

En primer lugar es necesario desarrollar los algoritmos de cada uno de los filtros digitales por separado. El procedimiento es el mismo para cualquier filtro que se quiera implementar en el dominio digital, tal y como se ha explicado en el capítulo [1.4]. Se parte de las especificaciones deseadas del filtro en el dominio discreto ω (rad) [8], realizando a posteriori la transformación de frecuencias al dominio analógico Ω (rad/s) mediante la **transformada bilineal** [9] y [10]. De esta forma se obtienen las especificaciones del filtro en el dominio analógico. A continuación se obtiene mediante la transformada de Laplace la función de transferencia H(s), y aplicando de nuevo la transformada bilineal se llega a la función de transferencia discreta H(z).

$$\omega_c = \frac{f_c}{f_s} 2\pi \quad (8)$$

$$\Omega_c = \frac{2}{T_s} tg \frac{\omega_c}{2} \quad (9)$$

$$s = \frac{2}{T_s}\frac{1 - z^{-1}}{1 + z^{-1}} \quad (10)$$

a) **Ecualizador tipo *Low Shelving*:** La función de transferencia en el dominio analógico de un ecualizador *Low Shelving* de 2º orden sigue la expresión siguiente, con A^2 la ganancia lineal del ecualizador, Ω_c la frecuencia de corte analógica y Q el factor de calidad [ref. 18].

$$H(s) = \frac{s^2 + \sqrt{2}(\sqrt{A}\Omega_c)s + (\sqrt{A}\Omega_c)^2}{s^2 + \sqrt{2}\left(\frac{\Omega_c}{\sqrt{A}}\right)s + \left(\frac{\Omega_c}{\sqrt{A}}\right)^2} = \frac{\left(\frac{s}{\Omega_c}\right)^2 + \sqrt{2}\sqrt{A}\left(\frac{s}{\Omega_c}\right) + A}{\left(\frac{s}{\Omega_c}\right)^2 + \frac{\sqrt{2}}{\sqrt{A}}\left(\frac{s}{\Omega_c}\right) + \frac{1}{A}} \quad (11)$$

Esta función de transferencia se corresponde con el diagrama de polos y ceros de la figura [24]. Los polos y ceros de esta función se encuentran en el mismo ángulo en circunferencias concéntricas, separados π/2 radianes. Al ser una función de segundo orden presenta dos polos y dos ceros en cada uno de los semiplanos. En función de la ganancia A, si A>1 los polos se acercan al eje jΩ, estando más cerca de este eje que los ceros. Por el contrario, si A<1 los ceros se acercan al eje jΩ. Para A=1, polos y ceros se anulan, por lo que el ecualizador no produce ningún realce o atenuación.

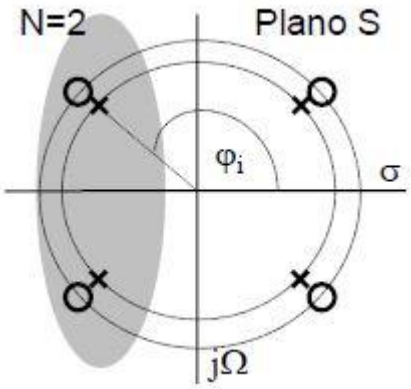

Figura 24: Diagrama de polos y ceros de un filtro Low Shelving.

Sustituyendo en la función de transferencia s/Ω_c según la transformada bilineal [12] se obtiene la función de transferencia en el dominio discreto H(z). Esta función cumple con la forma de la ecuación [13], mediante la cual pueden obtenerse los valores de los coeficientes del filtro en función de las especificaciones en el dominio discreto.

$$\frac{s}{\Omega_c} = \frac{\frac{2}{T_s}\frac{1 - z^{-1}}{1 + z^{-1}}}{\frac{2}{T_s}tg\left(\frac{\omega_c}{2}\right)} = \frac{\frac{1 - z^{-1}}{1 + z^{-1}}}{tg\left(\frac{\omega_c}{2}\right)} \quad (12)$$

$$H\,(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} = H\,(s)\Big|_{\frac{s}{\Omega_c} = \frac{\frac{1-z^{-1}}{1+z^{-1}}}{tg\left(\frac{\omega_c}{2}\right)}} \tag{13}$$

Operando la expresión anterior, la función de transferencia discreta es:

(14)

$$H(z) = \\ = \frac{\left(1 + cos(\omega_c)\right)(1 - 2z^{-1} + z^{-2}) + \sqrt{2}\sqrt{A}sen(\omega_c)(1 - z^{-2})}{\left(1 + cos(\omega_c)\right)(1 - 2z^{-1} + z^{-2}) + \sqrt{2}\frac{sen(\omega_c)}{\sqrt{A}}(1 - z^{-2}) + \frac{\left(1 - cos(\omega_c)\right)(1 + 2z^{-1} + z^{-2})}{A}} \\ + \frac{A\left(1 - cos(\omega_c)\right)(1 + 2z^{-1} + z^{-2})}{\left(1 + cos(\omega_c)\right)(1 - 2z^{-1} + z^{-2}) + \sqrt{2}\frac{sen(\omega_c)}{\sqrt{A}}(1 - z^{-2}) + \frac{\left(1 - cos(\omega_c)\right)(1 + 2z^{-1} + z^{-2})}{A}}$$

El diagrama de polos y ceros en el plano Z de esta función se muestra en la figura [25]. Siendo el ángulo la pulsación discreta ω (rad), en el caso de atenuación los ceros están más cerca de la circunferencia de radio unidad en el entorno de 0 radianes (bajas frecuencias), mientras que en el caso de realce son los polos los más cercanos.

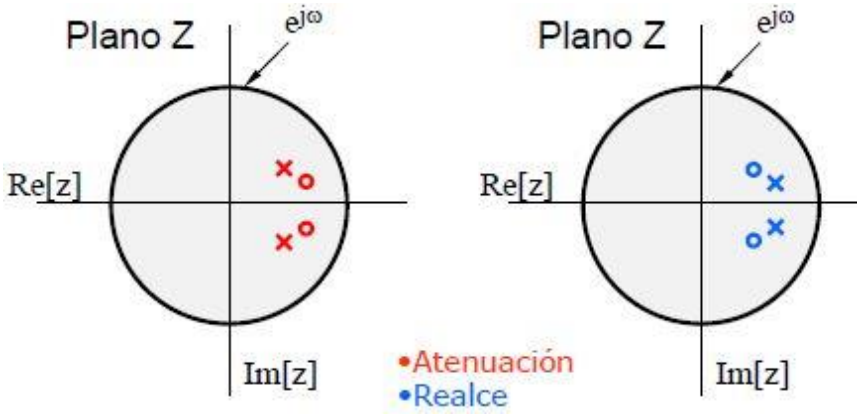

Figura 25: Diagrama de polos y ceros en el plano z de un filtro Low Shelving.

Identificando coeficientes en la función de transferencia discreta:

$$b_0 = (1 + A) + (1 - A)cos(\omega_c) + \sqrt{2}\sqrt{A}sen(\omega_c) \tag{15}$$

$$b_1 = -2[(1-A) + (1+A)cos(\omega_c)] \quad (16)$$

$$b_2 = (1+A) + (1-A)cos(\omega_c) - \sqrt{2}\sqrt{A}sen(\omega_c) \quad (17)$$

$$a_0 = \left(1+\frac{1}{A}\right) + \left(1-\frac{1}{A}\right)cos(\omega_c) + \frac{\sqrt{2}sen(\omega_c)}{\sqrt{A}} \quad (18)$$

$$a_1 = -2\left[\left(1-\frac{1}{A}\right) + \left(1+\frac{1}{A}\right)cos(\omega_c)\right] \quad (19)$$

$$a_2 = \left(1+\frac{1}{A}\right) + \left(1-\frac{1}{A}\right)cos(\omega_c) - \frac{\sqrt{2}sen(\omega_c)}{\sqrt{A}} \quad (20)$$

Estas expresiones de los coeficientes para un determinado valor de la ganancia y la frecuencia permiten construir el ecualizador en Matlab y realizar el filtrado de cualquier señal mediante la función *filter*. Cabe destacar la normalización del numerador y el denominador con respecto al término a_0, de forma que el término independiente del denominador sea igual a 1. A continuación se presenta la función en Matlab que entrega dos vectores **b** y **a** con los valores de cada uno de los coeficientes.

```
function [b,a] = LowShelving(w, dBgain)
%This function generate the coefficients for a second-order Low Shelving
%filter
%w: Cutoff Frequency
%dBgain: Gain in dB of the filter at w
A = sqrt(10^(dBgain/20));

b = zeros(1,3);%num coeff
a = zeros(1,3);%den coeff


a0 = (1+1/A) + (1-1/A)*cos(w)+ sqrt(2)*sin(w)/sqrt(A);  %Calculo de a0


b(1) = ((1 + A) + (1 - A)*cos(w)+ sqrt(2)*sqrt(A)*sin(w))/a0;
b(2) =( -2*( (1-A) +(1+A)*cos(w)))/a0;
b(3) = ((1 + A) + (1 - A)*cos(w) - sqrt(2)*sqrt(A)*sin(w))/a0;
a(1) = 1;%normalization
a(2) = (-2*((1-1/A) +(1+1/A)*cos(w)))/a0;
a(3) =  ((1 + 1/A) + (1 - 1/A)*cos(w)- sqrt(2)*sin(w)/sqrt(A))/a0;
```

La figura [26] muestra los resultados en Matlab obtenidos mediante la función *fvtool*, para una frecuencia discreta de ω = π/100 y una ganancia de +12dB. A su vez se muestra el diagrama de polos y ceros en el plano Z, el cual cumple con lo establecido previamente.

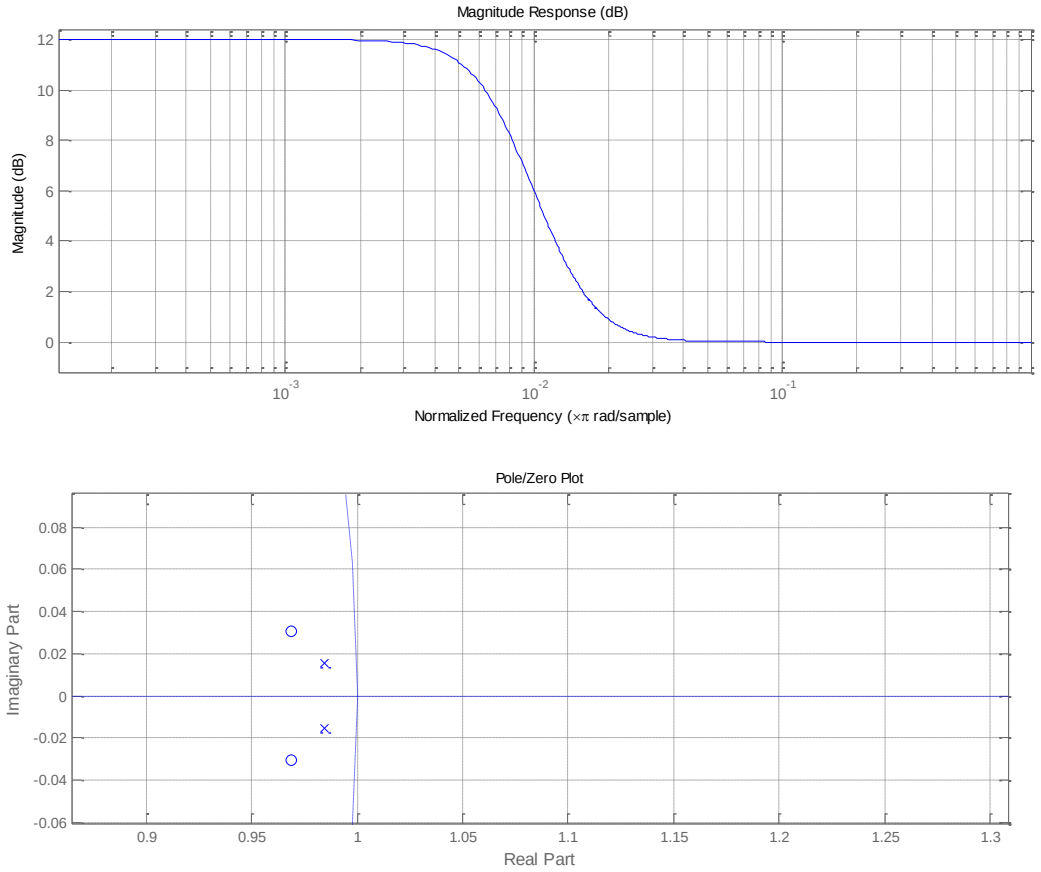

Figura 26: Filtro Low Shelving de frecuencia π/100 y ganancia 12dB (arrriba). Diagrama de polos y ceros (zoom) en el plano Z (abajo).

b) Ecualizador *High Shelving*: La función de transferencia en el dominio analógico de un ecualizador *High Shelving* de 2º orden se corresponde con la siguiente expresión, siendo la inversa de la función de transferencia de un ecualizador *Low Shelving*, multiplicada por la ganancia lineal del filtro A^2 [ref. 18].

$$H(s)_{High\ Shelving} = H(s)^{-1}_{Low\ Shelving} \cdot A^2 = A^2 \frac{\left(\frac{s}{\Omega_c}\right)^2 + \frac{\sqrt{2}}{\sqrt{A}}\left(\frac{s}{\Omega_c}\right) + \frac{1}{A}}{\left(\frac{s}{\Omega_c}\right)^2 + \sqrt{2}\sqrt{A}\left(\frac{s}{\Omega_c}\right) + A} \quad (21)$$

El diagrama de polos y ceros presenta una forma idéntica al de un ecualizador *Low Shelving*, con las mismas características de polos y ceros. Cuando A >1, los polos están más cerca del eje jΩ a una frecuencia mayor que los ceros. Cuando A<1, son los ceros los más cercanos al eje jΩ. Si A=0, polos y ceros se anulan.

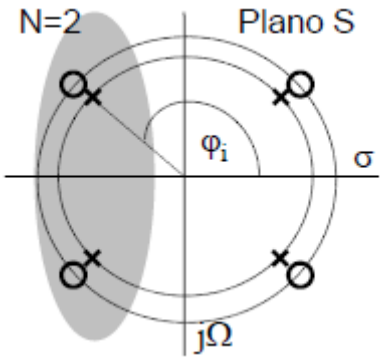

Figura 27: Diagrama de polos y ceros de un filtro High Shelving.

De nuevo aplicando la transformada bilineal se obtiene la función de transferencia en el dominio discreto, que cumple con la expresión siguiente:

(22)

$$H(z) = A^2 \left[\frac{\left(1+cos(\omega_c)\right)(1-2z^{-1}+z^{-2}) + \sqrt{2}\frac{sen(\omega_c)}{\sqrt{A}}(1-z^{-2})}{\left(1+cos(\omega_c)\right)(1-2z^{-1}+z^{-2}) + \sqrt{2}\sqrt{A}sen(\omega_c)(1-z^{-2}) + A\left(1-cos(\omega_c)\right)(1+2z^{-1}+z^{-2}} + \frac{\frac{\left(1-cos(\omega_c)\right)(1+2z^{-1}+z^{-2})}{A}}{\left(1+cos(\omega_c)\right)(1-2z^{-1}+z^{-2}) + \sqrt{2}\sqrt{A}sen(\omega_c)(1-z^{-2}) + A\left(1-cos(\omega_c)\right)(1+2z^{-1}+z^{-2})} \right]$$

El diagrama de polos y ceros en el plano Z se muestra en la figura [28]. Este diagrama es similar al de un ecualizador *Low Shelving* a la inversa. En el caso de realce, los polos se encuentran más de la circunferencia de radio unidad en el entorno de π radianes (altas frecuencias), mientras que en el caso de atenuación son los ceros los más cercanos.

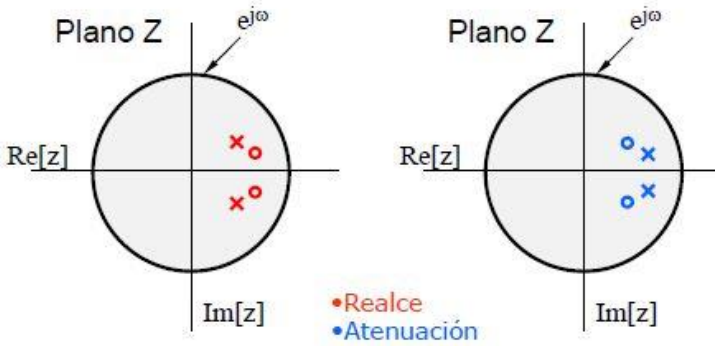

Figura 28: Diagrama de polos y ceros en el plano Z de un filtro High Shelving.

Identificando coeficientes mediante la expresión [13]:

$$b_0 = A^2\left[\left(1+\frac{1}{A}\right)+\left(1-\frac{1}{A}\right)cos(\omega_c)+\frac{\sqrt{2}sen(\omega_c)}{\sqrt{A}}\right] \quad (23)$$

$$b_1 = -2A^2\left[\left(1-\frac{1}{A}\right)+\left(1+\frac{1}{A}\right)cos(\omega_c)\right] \quad (24)$$

$$b_2 = A^2\left[\left(1+\frac{1}{A}\right)+\left(1-\frac{1}{A}\right)cos(\omega_c)-\frac{\sqrt{2}sen(\omega_c)}{\sqrt{A}}\right] \quad (25)$$

$$a_0 = (1+A)+(1-A)cos(\omega_c)+\sqrt{2}\sqrt{A}sen(\omega_c) \quad (26)$$

$$a_1 = -2[(1-A)+(1+A)cos(\omega_c)] \quad (27)$$

$$a_2 = (1+A)+(1-A)cos(\omega_c)-\sqrt{2}\sqrt{A}sen(\omega_c) \quad (28)$$

Se presenta a continuación la función de Matlab que permite obtener el valor de estos coeficientes en función de la frecuencia de corte y la ganancia del ecualizador.

```
function [b,a] = HighShelving(w, dBgain)
%This function generate the coefficients for a second-order High
Shelving
%filter
%w: Cutoff Frequency
%dBgain: Gain in dB of the filter at w
A = sqrt(10^(dBgain/20));

b = zeros(1,3);%num coeff
a = zeros(1,3);%den coeff

a0 = (1+A) + (1-A)*cos(w)+ sqrt(2)*sin(w)*sqrt(A);  %Calculo de a0

b(1) = (A^2*((1 + 1/A) + (1 - 1/A)*cos(w)+ sqrt(2)*sin(w)/sqrt(A)))/a0;
b(2) =(-2*A^2*((1-1/A) +(1+1/A)*cos(w)))/a0;
b(3) = (A^2*((1 + 1/A) + (1 - 1/A)*cos(w)- sqrt(2)*sin(w)/sqrt(A)))/a0;
a(1) = 1;%normalization
a(2) = (-2*((1-A) +(1+A)*cos(w)))/a0;
a(3) =  ((1 + A) + (1 - A)*cos(w)- sqrt(2)*sin(w)*sqrt(A))/a0;
```

La figura [29] muestra los resultados en Matlab obtenidos mediante la función *fvtool*, para una frecuencia discreta de ω = π/2 y una ganancia de +12dB. A su vez se muestra el diagrama de polos y ceros en el plano Z.

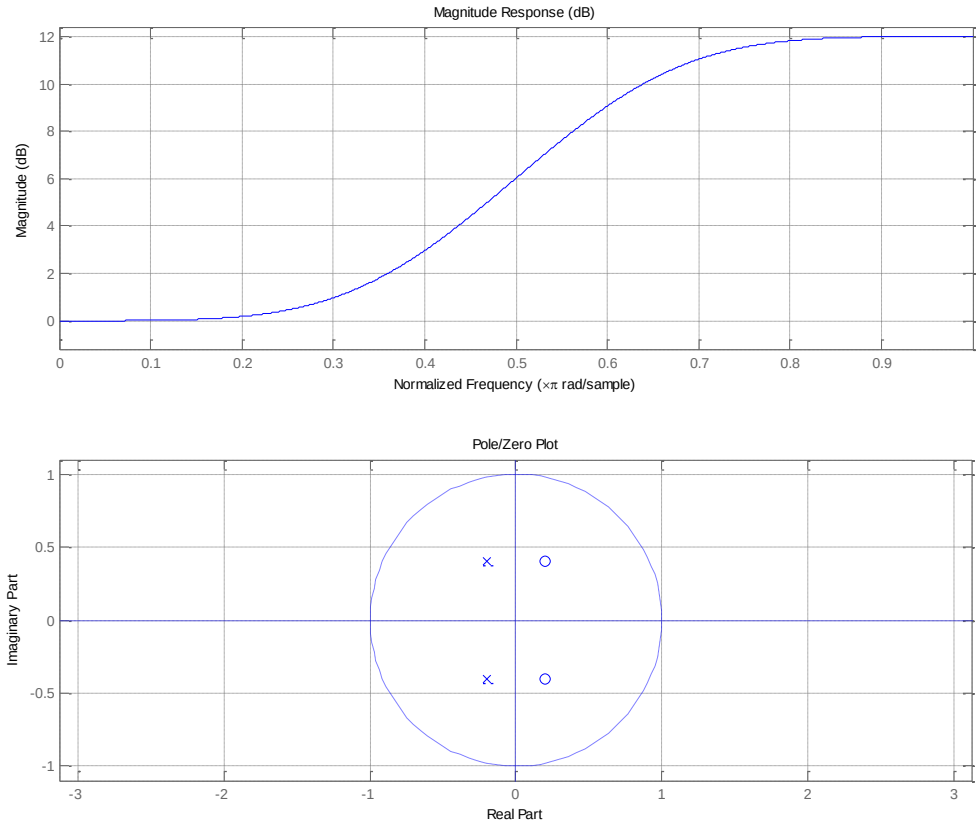

Figura 29: Filtro High Shelving de frecuencia π/100 y ganancia 12dB (arrriba). Diagrama de polos y ceros en el plano Z (abajo).

c) **Ecualizador resonante o ecualizador *Peak*:** La función de transferencia de un ecualizador *Peak* de 2º orden se corresponde con la expresión [31], siendo A la ganancia lineal de ecualizador y Q el factor de calidad [ref. 18]. Se define el factor de calidad y la frecuencia central según las expresiones siguientes:

$$Q = \frac{\Omega_c}{\Omega_2 - \Omega_1} \quad (29)$$

$$\Omega_c{}^2 = \Omega_2 \Omega_1 \quad (30)$$

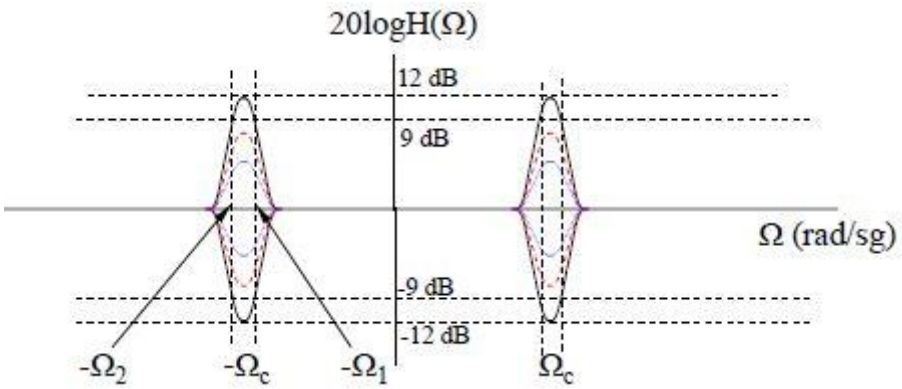

Figura 30: Especificaciones en el dominio analógico de un filtro Peak.

$$H(s) = \frac{s^2 + \frac{\sqrt{A}\Omega_c}{Q/2}s + \Omega_c^2}{s^2 + \frac{\Omega_c}{\sqrt{A}Q/2}s + \Omega_c^2} = \frac{\left(\frac{s}{\Omega_c}\right)^2 + \frac{\sqrt{A}}{Q/2}\left(\frac{s}{\Omega_c}\right) + 1}{\left(\frac{s}{\Omega_c}\right)^2 + \frac{1}{\sqrt{A}Q/2}\left(\frac{s}{\Omega_c}\right) + 1} \quad (31)$$

El diagrama de polos y ceros se muestra en la figura [31]. Presenta dos polos y dos ceros en cada uno de los semiplanos, separados π/2 radianes y en circunferencias concéntricas. En función de la ganancia A, si A>1 los polos se acercan al eje jΩ y el filtro produce realce. Si A<1, los ceros se acercan a jΩ y el ecualizador produce atenuación. Si A=1, polos y ceros se anulan y el ecualizador no produce ningún efecto.

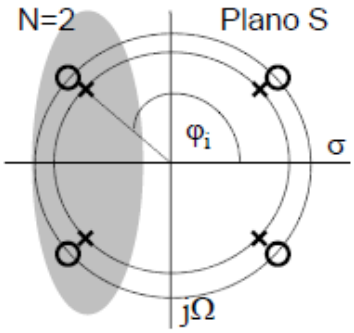

Figura 31: Diagrama de polos y ceros de un filtro Peak.

Aplicando la transformada bilineal se obtiene la función de transferencia discreta H(z), cuya expresión se muestra a continuación. La figura [32] muestra el diagrama de polos y ceros en el plano Z. Los polos se encuentran a la frecuencia de resonancia, mientras que los ceros aparecen más cerca que los polos de la circunferencia unidad en el caso de atenuación y más lejos en el caso de realce.

$$H(z) = \frac{\left(2 + \frac{\sqrt{A}sen(\omega_c)}{Q/2}\right) - 4cos(\omega_c)z^{-1} + \left(2 - \frac{\sqrt{A}sen(\omega_c)}{Q/2}\right)z^{-2}}{\left(2 + \frac{sen(\omega_c)}{\sqrt{A}Q/2}\right) - 4cos(\omega_c)z^{-1} + \left(2 - \frac{sen(\omega_c)}{\sqrt{A}Q/2}\right)z^{-2}} \quad (32)$$

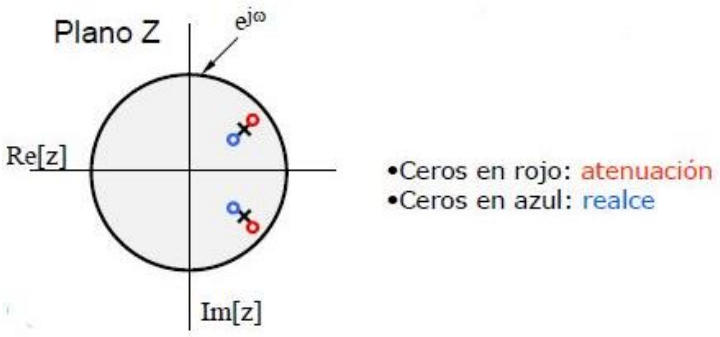

Figura 32: Diagrama de polos y ceros en el plano z de un filtro peak.

Identificando coeficientes mediante la expresión [13]:

$$b_0 = 2 + \frac{\sqrt{A}sen(\omega_c)}{Q} \quad (33)$$

$$b_1 = -4cos(\omega_c) \quad (34)$$

$$b_1 = 2 - \frac{\sqrt{A}sen(\omega_c)}{Q} \quad (35)$$

$$a_0 = 2 + \frac{sen(\omega_c)}{\sqrt{A}Q} \quad (36)$$

$$a_1 = -4cos(\omega_c) \quad (37)$$

$$a_2 = 2 - \frac{sen(\omega_c)}{\sqrt{A}Q} \quad (38)$$

Se presenta a continuación la función de Matlab que permite obtener el valor de estos coeficientes en función de la frecuencia de corte, la ganancia del ecualizador y el factor de calidad.

```
function [b,a] = Peak(w,dBgain,Q)
%This function generate the coefficients for a second-order Peak filter
%w: Resonant Frequency
%dBgain: Gain in dB of the filter at w
%Q: Quality factor
A = 10^(dBgain/20);

b = zeros(1,3);%Num coeff
a = zeros(1,3);%Den coeff

Q= Q/2;

a0 = 2+ (sin(w)/(sqrt(A)*Q));

b1= 2 + ((sqrt(A)*sin(w))/Q); b(1) = b1 /a0;
b2= -4*cos(w); b(2)= b2/a0;
b3 = 2 - ((sqrt(A)*sin(w))/Q); b(3) = b3/a0;
a(1) = 1;%normalization
a2= -4*cos(w);a(2) = a2 /a0;
a3= 2 - (sin(w)/(sqrt(A)*Q)); a(3) = a3/a0;
```

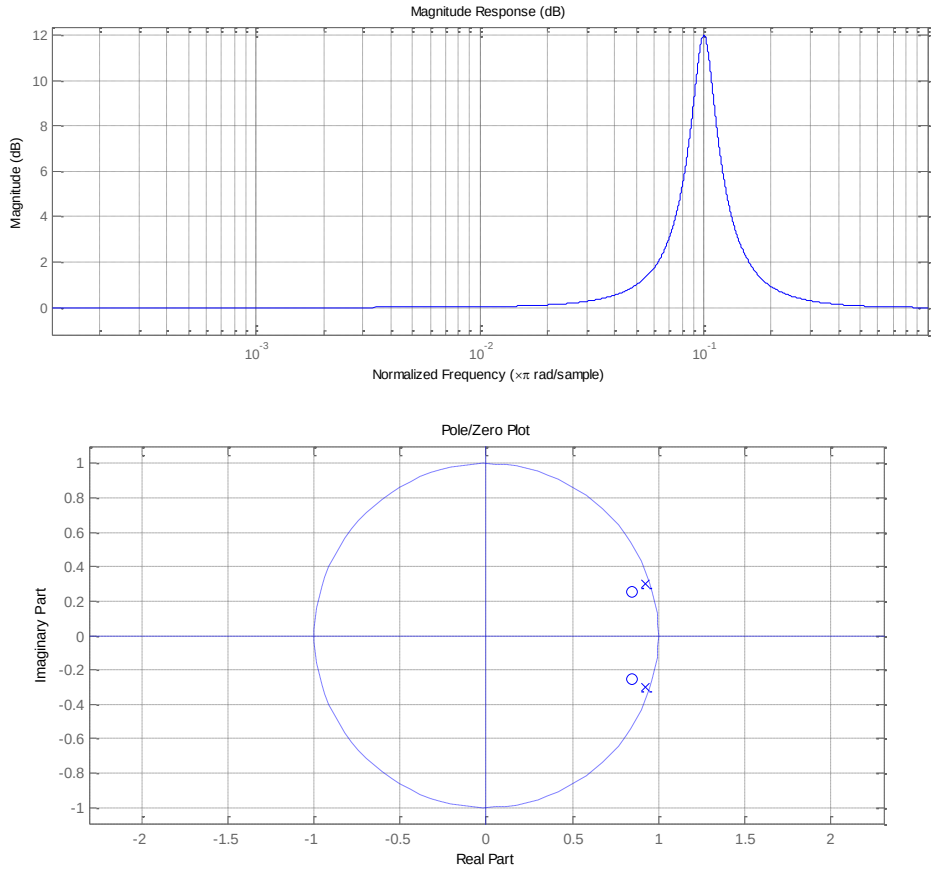

Figura 33: Filtro Peak de frecuencia π/10 y ganancia 12dB (arrriba). Diagrama de polos y ceros en el plano Z (abajo).

3.2.2. DESARROLLO DEL ECUALIZADOR PARAMÉTRICO EN SIMULINK

Como se ha explicado previamente, el ecualizador paramétrico ha sido construido mediante cinco etapas de ecualización en cascada. Simulink dispone de una amplia biblioteca potente para diseño de filtros digitales, con control de un gran número de parámetros que permiten un diseño preciso. Estos bloques se encuentran en la bilioteca de *DSP System Toolboox/Filtering*. Sin embargo, para un mayor conocimiento del sistema a implementar, se ha optado por generar los coeficientes mediante las funciones de Matlab desarrolladas anteriormente. Estas funciones generan dos vectores con los valores de los coeficientes del numerador y del denominador, los cuales son introducidos como datos de entrada en el bloque específico ***Digital Filter***.

La figura [34] muestra la ventana de configuración de este bloque. Seleccionando la opción *Input Port(s)* el bloque toma los coeficientes del filtro a partir de dos vectores (arrays). Debe seleccionarse también el tipo de función de transferencia (IIR en este caso), y la estructura del filtro, la cual se ha optado por la forma directa II (canónica). La casilla *First denominator coefficient* =1 sirve para indicar que los coeficientes del denominador vienen normalizados con respecto al término a_0. Finalmente debe seleccionarse el tipo de procesado de la señal de entrada,es decir, si se trata de Frame based o Sample Based.

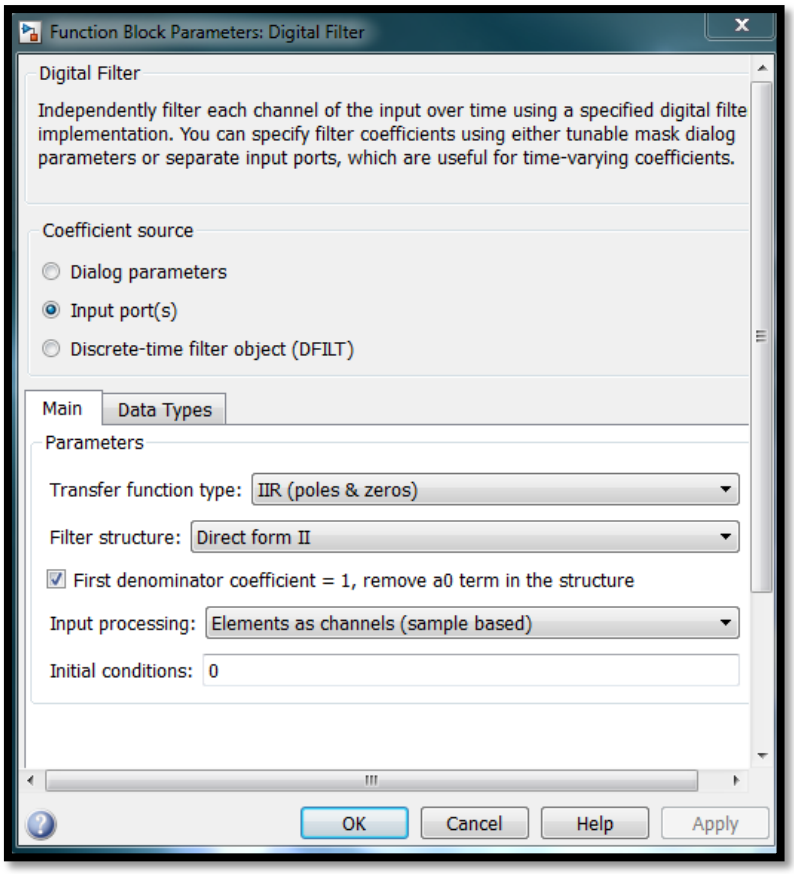

Figura 34: Ventana de configuración del bloque de Simulink Digital Filter.

Cada uno de los filtros se encuentra en un nivel inferior mediante un subsistema. Este subsistema incluye un bloque de función de Matlab con la función generadora de coeficientes. Además, se incluye un bloque de función matemática que transforma el parámetro de la frecuencia de entrada en Hz en frecuencia discreta en rad, siguiendo la expresión siguiente, con f_s=48kHz:

$$\omega_c(rad) = \frac{f_c}{f_s} 2\pi \tag{39}$$

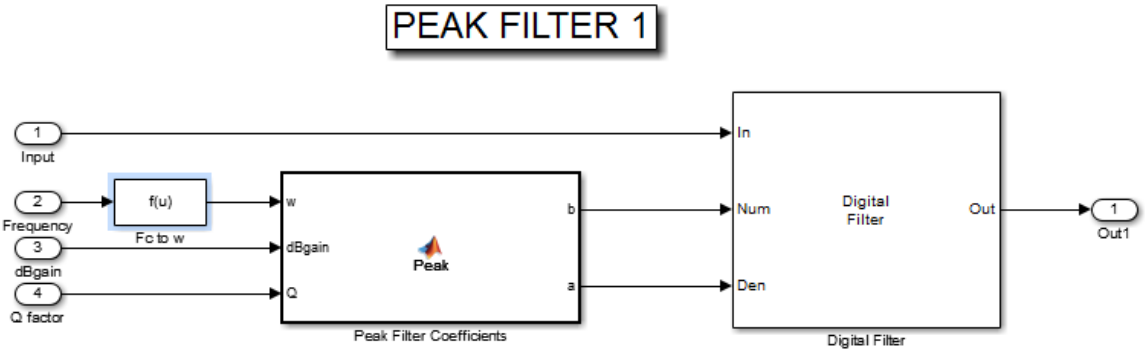

Figura 35: Subsistema de un filtro Peak del ecualizador paramétrico en Simulink.

La figura [36] muestra el diagrama del nivel superior en Simulink, donde cada uno de los bloques de constantes son los valores de los parámetros de cada uno de los filtros, que serán asignados a los potenciómetros de control del plugin. Un conmutador manual de ***Bypass*** al final de la cadena permite seleccionar entre la señal sin tratar y la señal filtrada. Los canales de entrada y de salida marcados con color azul son los correspondientes a la biblioteca de APG. La sección de visualización de la respuesta en frecuencia se ha obtenido mediante el **estimador de la función de transferencia**. Este analizador toma como señales de entrada la señal de ruido antes de filtrar y la señal filtrada. La señal de ruido ha sido concatenada con el canal de entrada mediante el bloque *matrix concatenate*, y ambas señales son separadas mediante el bloque *Selector*. El ruido ha sido configurado como **ruido blanco** con valores aleatorios entre -1 y 1, con un tiempo de muestra de 1/48000 s. Finalmente, a la salida del analizador se incluye el bloque *Access Point* para visualización de la respuesta en frecuencia en el plugin, y un *Vector Scope* en el dominio de la frecuencia que permite visualizar la respuesta en frecuencia en Simulink.

El analizador de la respuesta en frecuencia ha sido configurado con una FFT de 256 muestras de longitud. La visualización de la respuesta no requiere de una resolución alta ya que se trata de una estimación, y debe cambiar lo suficientemente rápido al realizar variaciones en los parámetros del ecualizador. Además se ha seleccionado una media espectral de 100 muestras. El rango de visualización ha sido seleccionado desde 0 hasta un valor inferior a Fs/2 (22kHz).

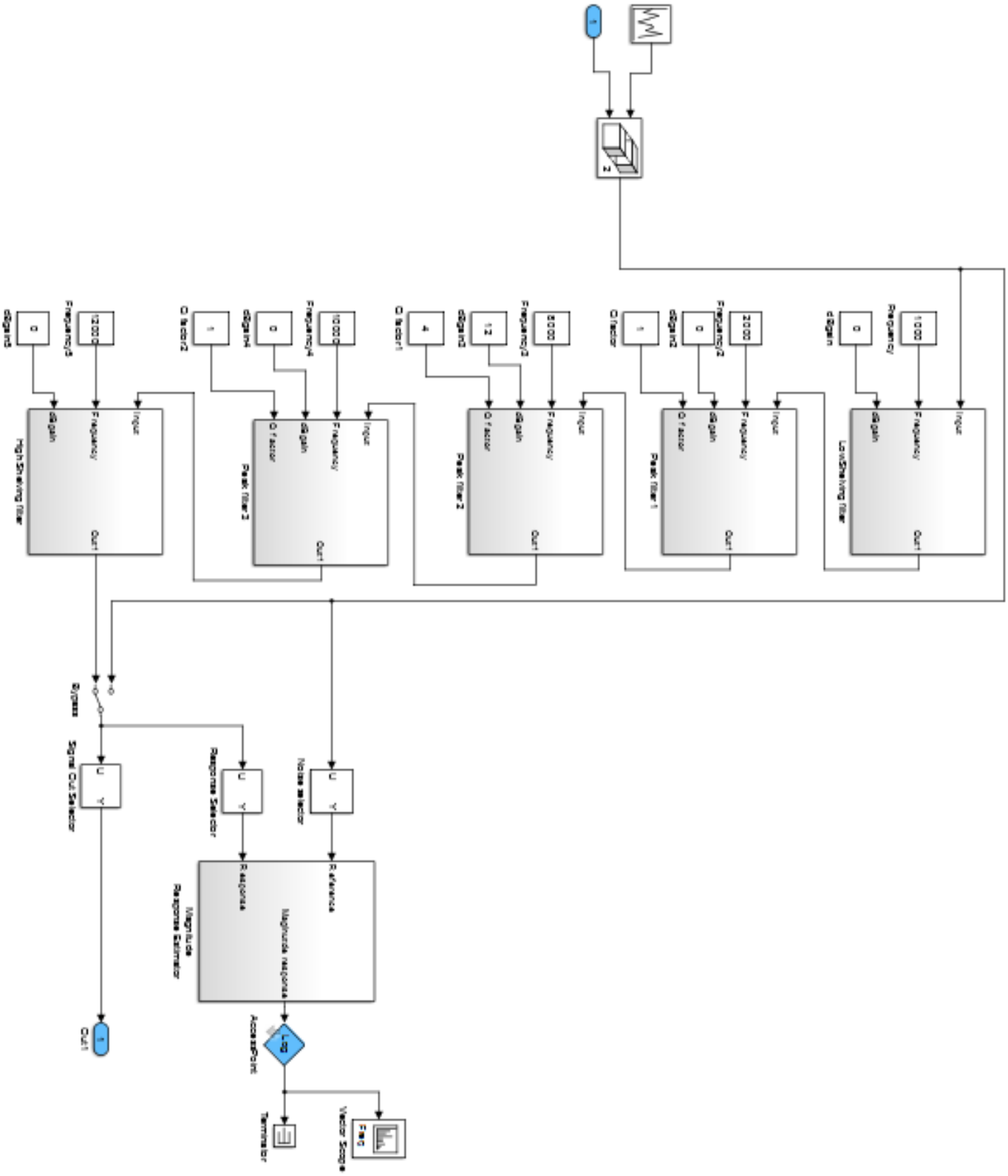

Figura 36: Diagrama de bloques en Simulink del ecualizador paramétrico

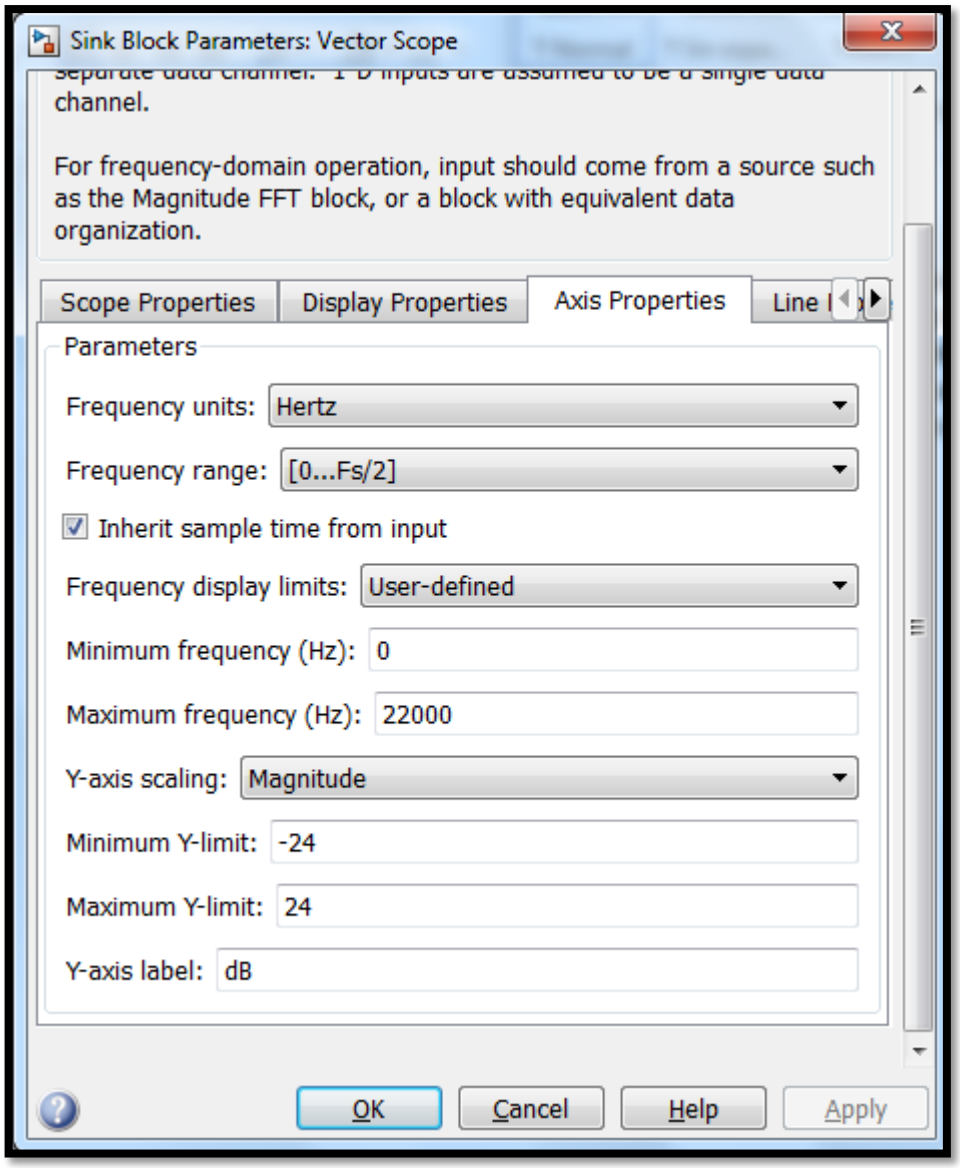

Figura 37: Ventana de configuración del bloque Vector Scope.

A continuación se muestra el resultado del ecualizador tras el analizador de la respuesta en frecuencia, mediante el *Vector Scope*. El ecualizador está configurado con una ganancia en los filtros de *Low Shelving* y *High Shelving* de +12 dB y frecuencias de corte de 500Hz y 15000Hz respectivamente. Se ha configurado también uno de los filtros *Peak* con un realce de +12 dB a la frecuencia de 8000Hz.

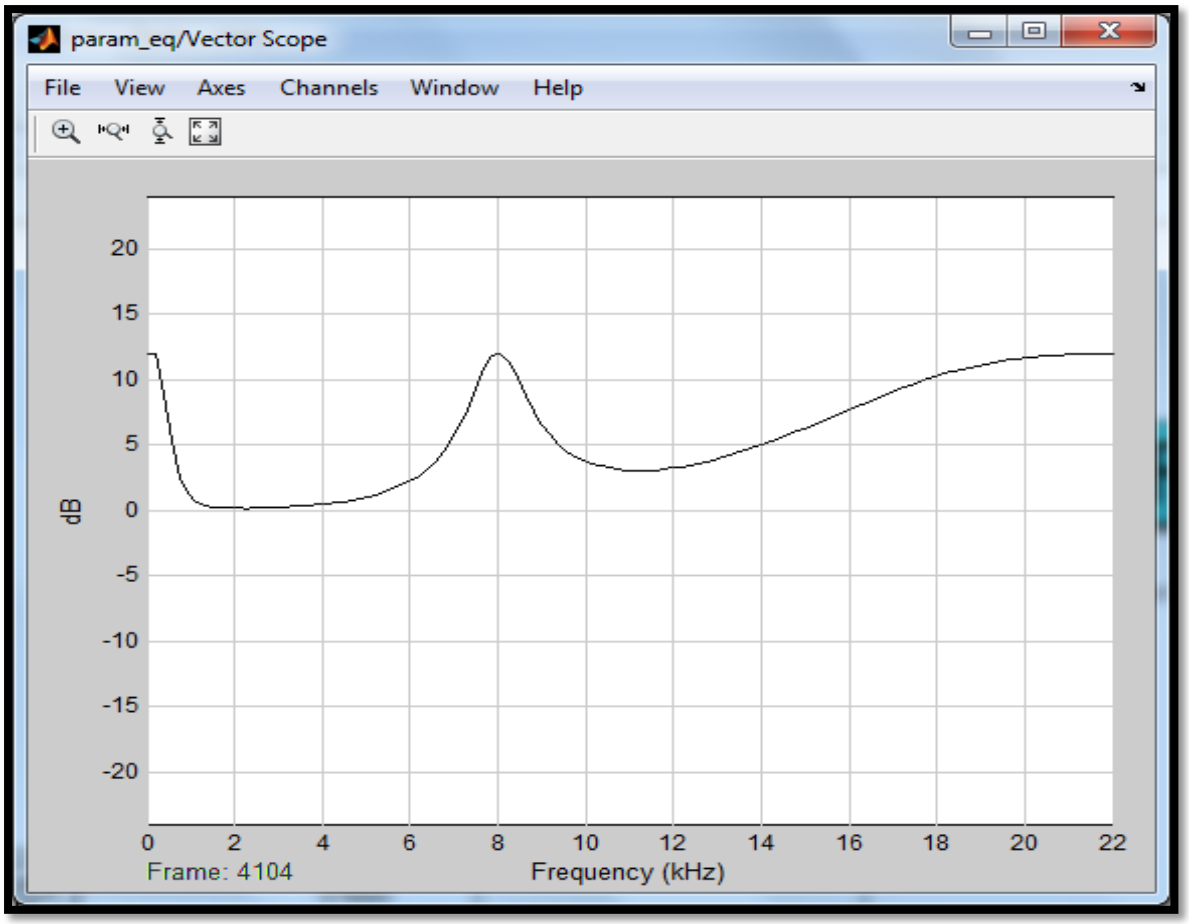

Figura 38: Respuesta en frecuencia del ecualizador paramétrico. Parámetros: Low Shelving: f1=500Hz. dBgain1=12dB; Peak1: f2= 8000Hz,dBgain2= 12dB, Q=7; High Shelving: f5= 15000Hz, dBgain5=12dB.

3.3. ECUALIZADOR GRÁFICO

Los ecualizadores gráficos cubren toda la banda de audio (20Hz-20kHz) mediante una serie de filtros paso banda. Dependiendo del número de bandas, se clasifican en ecualizadores de bandas de octava (10 bandas), ecualizadores de media octava (20 bandas) y ecualizadores de tercios de octava (30 bandas), siendo este último el más usado en el ámbito profesional. Las frecuencias centrales de las bandas están establecidas según la **norma ISO** [ref. 14], y se muestran en la tabla [1]. Los ecualizadores gráficos no poseen control del ancho de banda, siendo este factor fijo según el número de bandas. La expresión [40] permite calcular el factor de calidad en función del número de bandas.

$$Q = \frac{\sqrt{K}}{K-1} \quad \begin{cases} octavas: & K = 2^1 \\ medias\ octavas: & K = 2^{1/2} \\ tercios\ de\ octava: & K = 2^{1/3} \end{cases} \qquad (40)$$

*Tabla 1: Bandas de frecuencia según la norma ISO para octavas, medias octavas y tercios de octava. *La frecuencia de 18kHz no pertenece a la norma, sin embargo es utilizada frecuentemente por ecualizadores comerciales.*

Frec.	1	1/2	1/3	Frec.	1	1/2	1/3	Frec.	1	1/2	1/3
16	*	*	*	160			*	1.600			*
18				180		*		1.800			
20			*	200			*	2.000	*	*	*
22.4		*		224				2.240			
25			*	250	*	*	*	2.500			*
28				280				2.800		*	
31.5	*	*	*	315			*	3.150			*
35.5				355		*		3.550			
40			*	400			*	4.000	*	*	*
45		*		450				4.500			
50			*	500	*	*	*	5.000			*
56				560				5.600		*	
63	*	*	*	630			*	6.300			*
71				710		*		7.100			
80			*	800			*	8.000	*	*	*
90		*		900				9.000			
100			*	1.000	*	*	*	10.000			*
112				1.120				11.200		*	
125	*	*	*	1.250			*	12.500			*
140				1.400		*		14.000			
160			*	1.600			*	16.000	*	*	*
								18.000			(*)

Existen distintas formas de implementar un ecualizador gráfico, proceso que aunque parezca sencillo da lugar a un problema muy importante y objetivo aún de desarrollo, la interacción entre bandas. En este proyecto la implementación del ecualizador se ha logrado mediante la técnica de **filtrado FFT**. Se dan las pautas también para desarrollar un ecualizador gráfico mediante un **banco de filtros paso banda** o **ecualizadores tipo *Peak***, sin solucionar el problema de la interacción entre bandas explicado en los apartados [3.3.1] y [3.3.2].

3.3.1. ECUALIZADOR GRÁFICO MEDIANTE UN BANCO DE FILTROS PEAK EN PARALELO

La forma más intuitiva de desarrollar un ecualizador es la implementación de un banco de filtros paso banda o ecualizadores tipo *Peak* dispuestos en **paralelo**. La señal de entrada pasa por cada uno de los filtros, los cuales aplican la atenuación o realce deseada a su margen de frecuencias. A continuación se suman las señales de salida, obteniendo la señal filtrada en todo el margen de audio. Este proceso de suma provoca dos problemas difíciles de erradicar. En primer lugar, el factor de calidad fijo Q y, por tanto, el ancho de banda de cada uno de los filtros sólo se cumple para el caso de máxima atenuación o realce. Esto se debe a que en cualquier filtro, se mantiene constante el producto ganancia por ancho de banda. Por tanto, al reducir la atenuación o realce, disminuye el factor Q y el ancho de banda.

Por otro lado, las bandas de frecuencia de estos filtros están solapadas, tal y como muestra la figura [39]. Esta interacción se hace notable cuando se presentan un gran número de bandas cercanas entre sí, como es caso de utilizar tercios de octava, provocando que el ecualizador en su conjunto de filtros presente una ganancia mucho mayor a la deseada y dando lugar a una ecualización no deseada por el usuario.

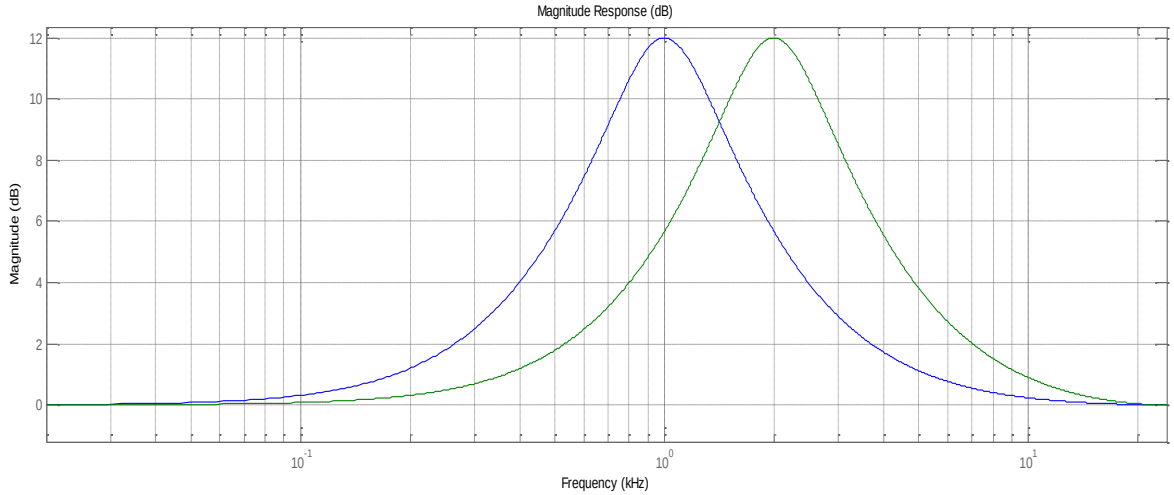

Figura 39: Respuesta en frecuencia de dos bandas adyacentes, con frecuencias centrales de 1kHz y 2 kHz. Posición de máximo realce. Q ≈ 1.4142.

Para comprender más a fondo el defecto de solape, se considera un ecualizador gráfico de 10 bandas, con todas las bandas dispuestas con una ganancia de 0dB (sin ecualizar). Esta ganancia en escala lineal se obtiene mediante la expresión [41], por lo que 0dB se corresponde con un valor de ganancia 1. Al sumar la salida de cada uno de los filtros en escala lineal, se produce una ganancia correspondiente con la expresión [42].

$$A = 10^{dBgain/20} \tag{41}$$

$$a_{total} = A_1 + A_2 + A_3 + \cdots + A_N \qquad donde\ N = nº\ bandas \tag{42}$$

Aplicando logaritmos a la expresión anterior:

$$A_{total}(dB) = 20log(a_{total}) = 20log(10^{\frac{a_1}{20}} + 10^{\frac{a_2}{20}} + 10^{\frac{a_3}{20}} + \cdots + 10^{\frac{a_N}{20}}) \tag{43}$$

Para todos los filtros configurados con ganancia 0dB, la ganancia por defecto que presenta el ecualizador se corresponde con la expresión [44].

$$A_{total}(dB) = 20log(N) \tag{44}$$

Para corregir este efecto la solución es aplicar una atenuación de valor N a la salida de cada filtro. Considerando a continuación una banda configurada en máximo realce (+12dB), en el punto de mayor amplitud el filtro presenta una ganancia lineal de valor 4. Aplicando a continuación la atenuación de valor N, en el caso de disponer de 10 bandas, la ganancia final en el punto de mayor amplitud tiene un valor de 20log(4/10+9/10) ≈ 2.27dB, muy lejos del valor deseado de ecualización. Por tanto, la solución propuesta es construir el filtro con una ganancia mayor a la deseada, ponderando la ganancia máxima por un factor **k**. De esta forma, tras aplicar a cada filtro su atenuación y la suma de los filtros restantes, el resultado final se corresponde con la ganancia deseada por el usuario.

Para calcular el factor k, basta con establecer la expresión [45], donde la ganancia deseada se divide por la atenuación de valor N y se suma junto con la interacción del resto de bandas.

$$a = \frac{a.k}{N} + \frac{N-1}{N} \tag{45}$$

Resolviendo la ecuación:

$$k = \frac{1-N}{a} + N \tag{46}$$

Esta solución se presenta incompleta, ya que la misma ponderación no puede ser utilizada para configuraciones de atenuación del ecualizador. En estos casos, el factor k toma un valor negativo, por lo que resulta imposible utilizar este método para construir filtros atenuantes. Además, al mantenerse constante el producto ganancia por ancho de banda, el resultado final de cada filtro que conforma el ecualizador es un ancho de banda inferior al establecido en la expresión [40]. La figura [40] muestra la salida del ecualizador construido en Simulink con todas las bandas en la posición de máximo realce.

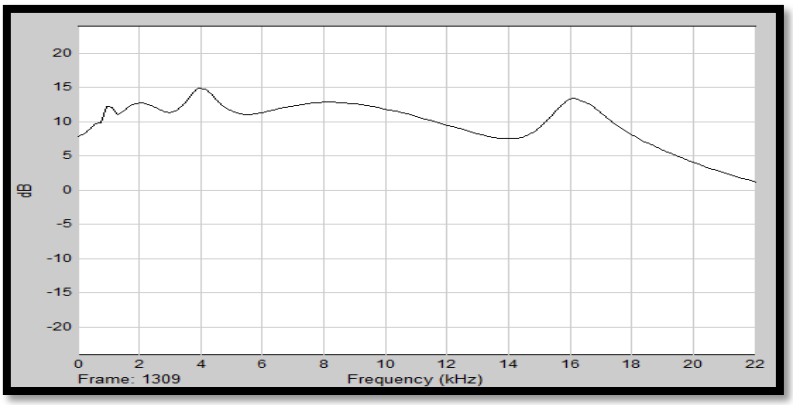

Figura 40: Ecualizador gráfico con 10 filtros Peak ponderados en paralelo. Todas las bandas en la posición de realce. $Q_{inicial} \approx 1{,}4142$.

3.3.2. ECUALIZADOR GRÁFICO MEDIANTE UN BANCO DE FILTROS PEAK EN CASCADA

Al igual que un ecualizador gráfico puede implementarse mediante la topología en paralelo, otra opción es realizar una topología de filtros dispuestos en **cascada**. Esta implementación evita la componente continúa añadida por el conjunto de filtros de la expresión [44] al no sumar las salidas. Sin embargo, la interacción producida por el solape entre bandas es mucho mayor, alcanzando el ecualizador elevadas ganancias (figura [41]). Para evitar esto, al mantenerse constante el producto de ganancia por ancho de banda, es necesario que el factor de calidad Q de cada filtro se multiplique por la ganancia lineal deseada. De esta forma se reduce proporcionalmente el ancho de banda de cada filtro, disminuyendo la interacción entre bandas. En el caso de atenuaciones, el factor de calidad Q se debe multiplicar por la inversa de la ganancia para reducir el ancho de banda, ya que la atenuación lineal tiene un valor menor a 1.

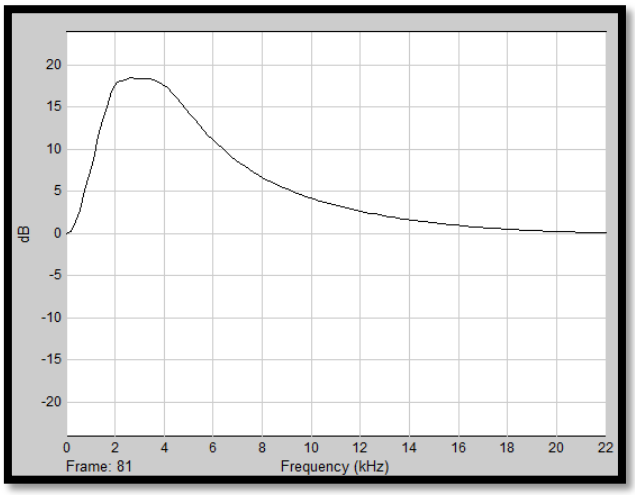

Figura 41: Respuesta en frecuencia de dos ecualizadores adyacentes tipo Peak en cascada, en la posición de máximo realce. Q ≈ 1,4142.

Este método de implementación de un ecualizador gráfico no está libre de errores. Si se visualiza la respuesta en frecuencia del ecualizador con todas las bandas dispuestas en la posición de máximo realce o atenuación, puede observarse un rizado de gran magnitud, efecto no deseado en el proceso de ecualización de audio. Este efecto se debe al uso de filtros con un ancho de banda mucho menor al deseado según el número de bandas. Por tanto, debe llegarse a una solución de compromiso entre obtener una mayor ganancia de la deseada o la magnitud del rizado producido por el conjunto de filtros.

La implementación en Simulink se muestra en la figura [42]. La estructura es prácticamente idéntica a la utilizada por el ecualizador paramétrico en el apartado [3.2]. Se disponen los filtros necesarios según el número de bandas, conectando la salida de cada uno a la entrada del siguiente. El estimador de la función de transferencia permite

visualizar la respuesta en frecuencia del ecualizador. La constante ***Q ponderation*** permite disminuir el factor de calidad Q, para así poder elegir entre un mayor o menor rizado y una ganancia superior o inferior a la deseada.

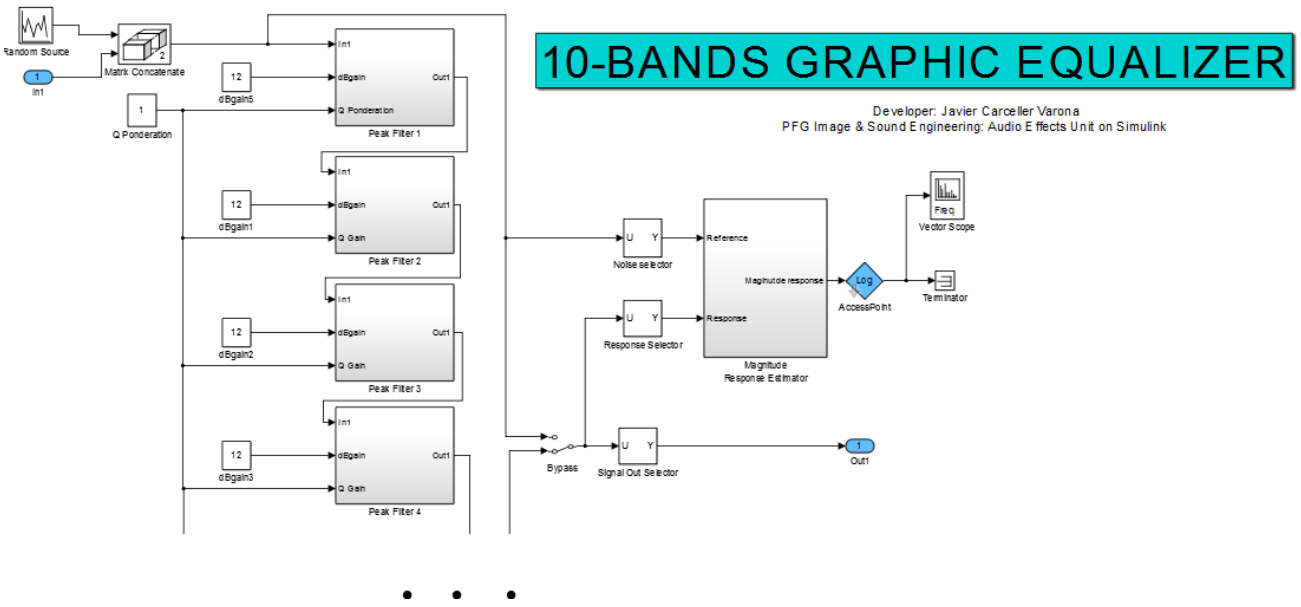

Figura 42: Estructura en Simulink de un ecualizador gráfico de 10 bandas mediante ecualizadores tipo Peak en cascada.

Para construir cada uno de los filtros se ha utilizado la estructura de ecualizadores tipo *Peak* del apartado [3.2]. En esta ocasión, la frecuencia y factor de calidad no son configurables, siendo este último únicamente modificado según lo establecido previamente según la estructura mostrada en las figuras [43] y [44]. Se utilizan los bloques de condición *if* para realizar la ponderación diferente en función de si la ganancia en dB es positiva o negativa.

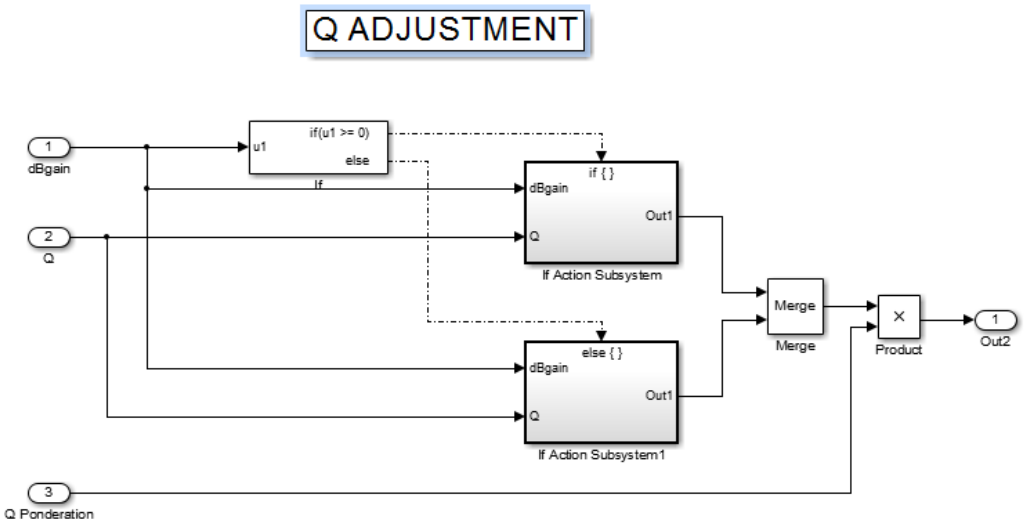

Figura 43: Estructura del bloque Q Adjustment.

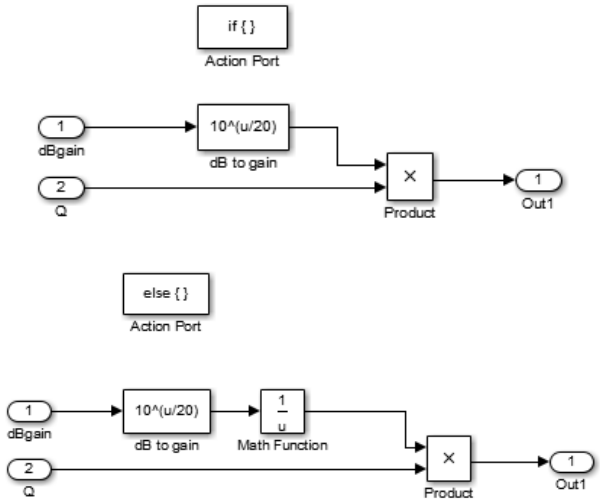

Figura 44: Ponderación del factor de calidad Q según la amplitud. Caso de ganancia (izquierda) y atenuación (derecha).

La figura [45] muestra la respuesta en frecuencia del ecualizador gráfico de 10 bandas, para todas las bandas configuradas en máximo realce y máxima atenuación.

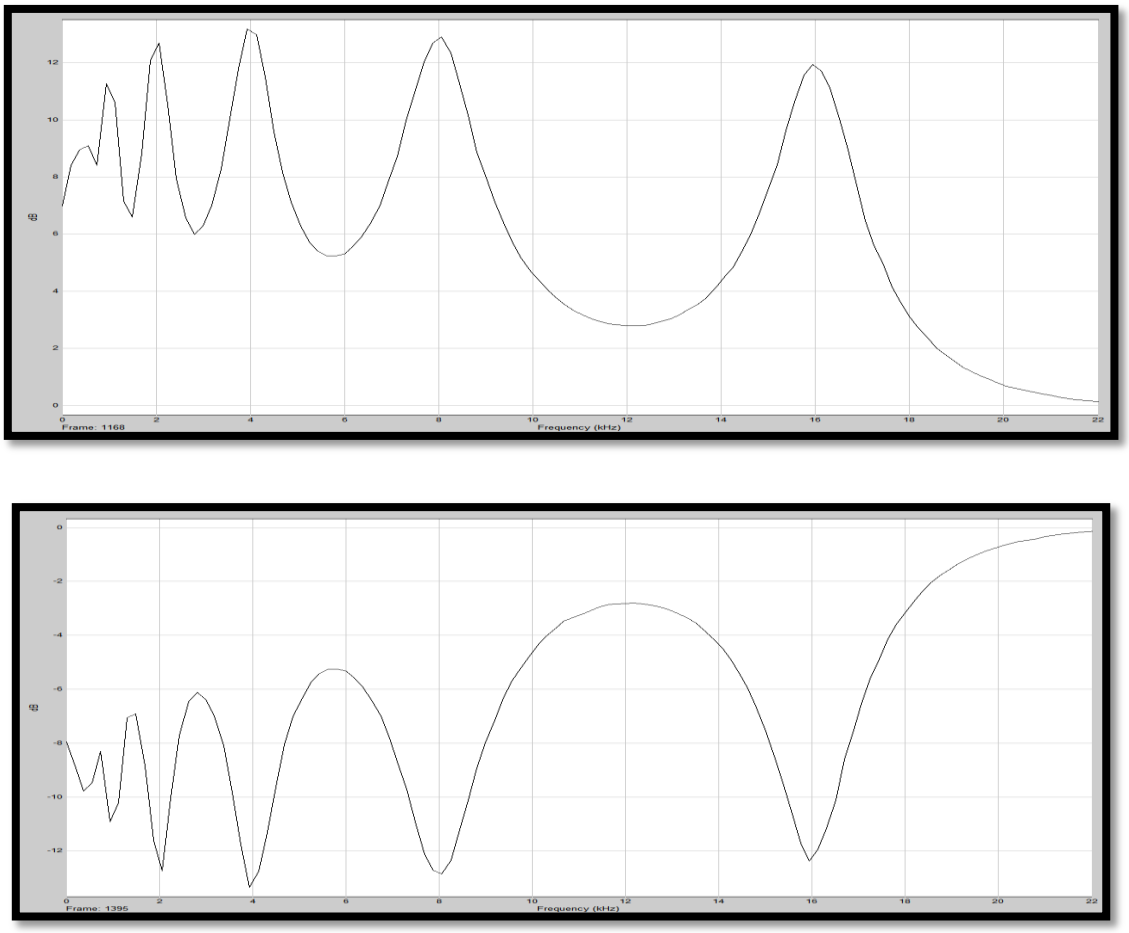

Figura 45: Respuesta en frecuencia del ecualizador gráfico de 10 bandas. Posición de máximo realce (arriba) y posición de máxima atenuación (abajo).

3.3.3. ECUALIZADOR GRÁFICO MEDIANTE FFT

El proceso de filtrado de una señal en frecuencia puede realizarse también gracias a la técnica de la transformada rápida de Fourier o ***FFT*** (*Fast Fourier Transform*). Mediante esta técnica, se utiliza la propiedad de la transformada de Fourier por la cual la convolución de dos señales en el tiempo se corresponde con su multiplicación en frecuencia. El proceso consiste en el diseño por muestras de la respuesta en frecuencia del ecualizador, dividiendo la respuesta en bandas según el número de muestras que corresponden a cada banda, y eligiendo la amplitud correspondiente a cada una de las muestras.

Para poder realizar la multiplicación en el espectro, es necesario procesar la señal de entrada mediante el método de análisis – síntesis. La señal de entrada es enventanada con un solape del 50%. A continuación se realiza la *FFT* para obtener la respuesta en frecuencia de cada segmento. Cada segmento se multiplica por la respuesta en frecuencia del ecualizador, realizando la reconstrucción de la señal en el proceso de síntesis mediante la transformada inversa de Fourier o *IFFT*. Se aplica de nuevo el enventanado a la señal y el método *Overlap–-add* [7.3.1] para la correcta reconstrucción, ya que el proceso de enventanado con solape modifica la frecuencia de muestreo de la señal.

Existen dos consideraciones fundamentales a tener en cuenta en este método. En primer lugar, dada una longitud de cada segmento de la señal de entrada de N_1 muestras, la convolución en el tiempo o la multiplicación en frecuencia con la respuesta del ecualizador de N_2 muestras genera una longitud de salida de N_1+N_2-1. Por tanto, para obtener la señal de salida de la longitud adecuada es necesario rellenar con ceros cada segmento de la señal de entrada y la respuesta en frecuencia del ecualizador para alcanzar la longitud anterior. De lo contrario, la señal de salida aparecería truncada.

En segundo lugar, para un correcto filtrado sin artificios de la señal de entrada es necesario utilizar un filtro de fase lineal. La respuesta en frecuencia de un filtro con fase lineal se corresponde con la expresión [47] [ref. 12], donde $A\left(e^{j\Omega}\right)$ es el módulo diseñado por muestras del ecualizador y N_F el número de muestras de la *FFT*.

$$H\left(e^{j\Omega}\right) = A\left(e^{j\Omega}\right)e^{-j\left(N_F-\frac{1}{2}\right)\Omega} \qquad (47)$$

Para el diseño del módulo, al tomar una *FFT* de longitud N_2 debe tenerse en cuenta que la respuesta a diseñar es simétrica, teniendo cada parte una longitud de $N_2/2$ muestras. El espectro debe repartirse de forma equitativa entre las N_2 muestras, cumpliendo con la expresión [48], la cual muestra la correspondencia entre cada frecuencia y su muestra correspondiente.

$$\frac{f}{f_s} = \frac{k}{N_F} \qquad con\ k = 0,1,\dots,N_F-1 \qquad (48)$$

Analizando la fase del filtro, se divide mediante la ecuación de *Euler* en parte real e imaginaria:

$$e^{-j\left(N_F-\frac{1}{2}\right)\Omega} = e^{-j\left(N_F-\frac{1}{2}\right)\frac{2\pi k}{N_F}} = cos\left(2\pi\frac{N_F-1}{2}\frac{k}{N_F}\right) - jsen\left(2\pi\frac{N_F-1}{2}\frac{k}{N_F}\right) \quad (49)$$

$$con \quad k = 0,1,\ldots,\frac{N_F}{2}-1$$

Ambas partes deben presentar una respuesta simétrica, por lo que al ser la función del ecualizador H(z) real y de longitud finita, debe cumplirse además la condición:

$$H(k=\frac{N_F}{2}) = 0 \quad (50)$$

La figura [46] muestra la implementación del ecualizador gráfico mediante *FFT*. El subsistema *Filter Response*, mostrado en la figura [47], se encarga de la construcción del filtro de fase lineal de longitud *NFFT* muestras. La salida de este subsistema es la respuesta en frecuencia del filtro, siendo ésta multiplicada por la respuesta en frecuencia de cada segmento de la señal de entrada, salida del subsistema *Analysis System*.

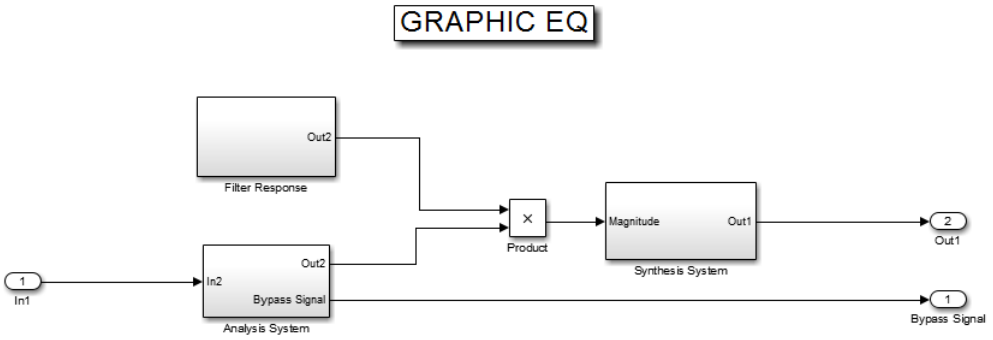

Figura 46: Ecualizador gráfico FFT.

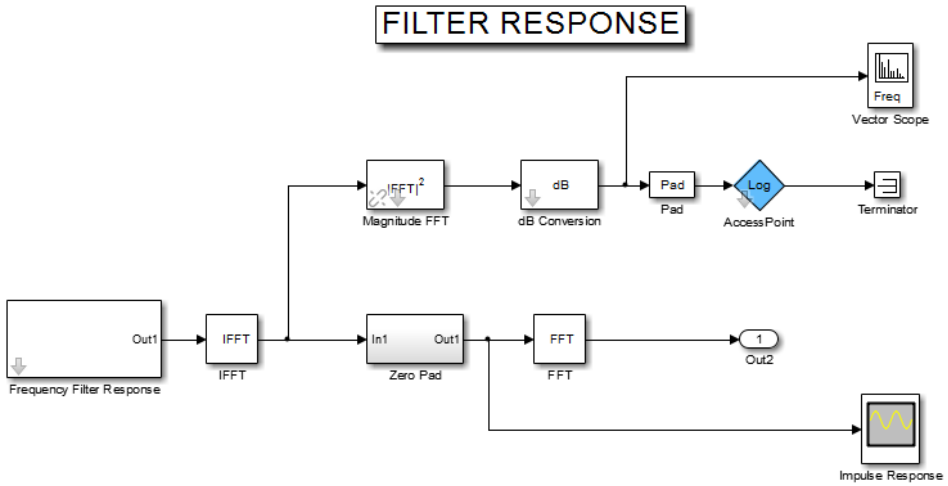

Figura 47: Subsistema Filter Response.

La rama superior del subsistema *Filter Response* permite visualizar la respuesta en frecuencia del ecualizador. La *FFT* devuelve la transformada de NFFT muestras de las cuales NFFT/2 son las muestras simétricas. Estas muestras se eliminan mediante el bloque *Pad,* ya que no se van a representar. Este proceso no es necesario en el bloque visualizador de la señal *Vector Scope*, ya que se selecciona un rango de visualización hasta *fs/2*. El bloque *Zero Pad* añade un número determinado de ceros para alcanzar la longitud $N_1 + N_2$ (Matlab trabaja con vectores cuyo primer índice es el 1, no el 0). La figura [48] muestra la estructura del bloque *Frequency Filter Response*, donde la matriz de ganancias crea un vector de NFFT/2 muestras con la configuración de amplitudes deseadas del ecualizador gráfico. El bloque *Linear Phase* crea dos vectores de longitudes NFFT/2 con la parte real e imaginaria de la fase cumpliendo con la expresión [49]. Se multiplica la magnitud del filtro por la parte real e imaginaria y se construye la mitad simétrica, concatenando ambas partes. Finalmente se unen en valores complejos mediante el bloque *Real-Imag to complex*, realizando a posteriori la transformada inversa.

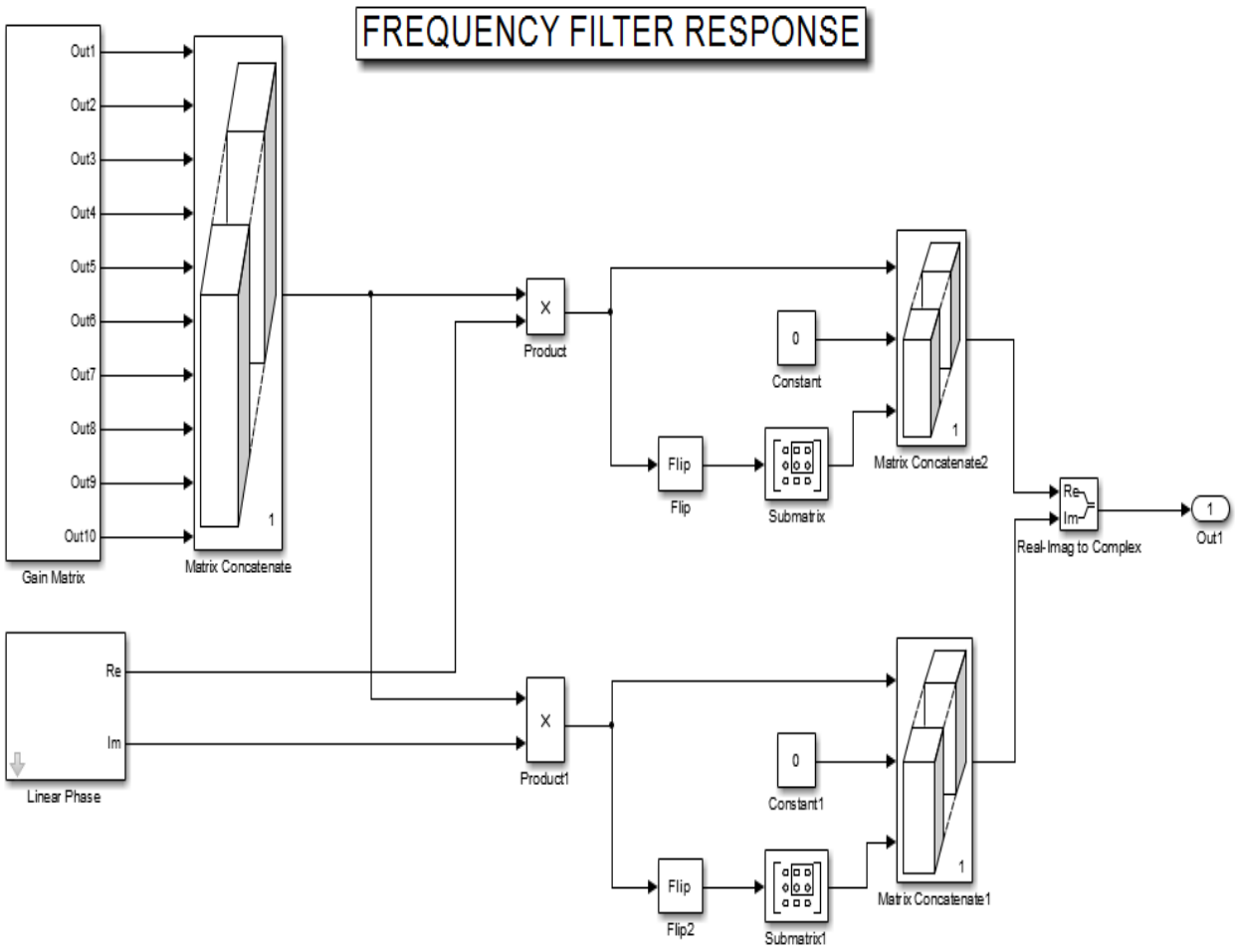

Figura 48: Subsistema Frequency Filter Response para 10 bandas.

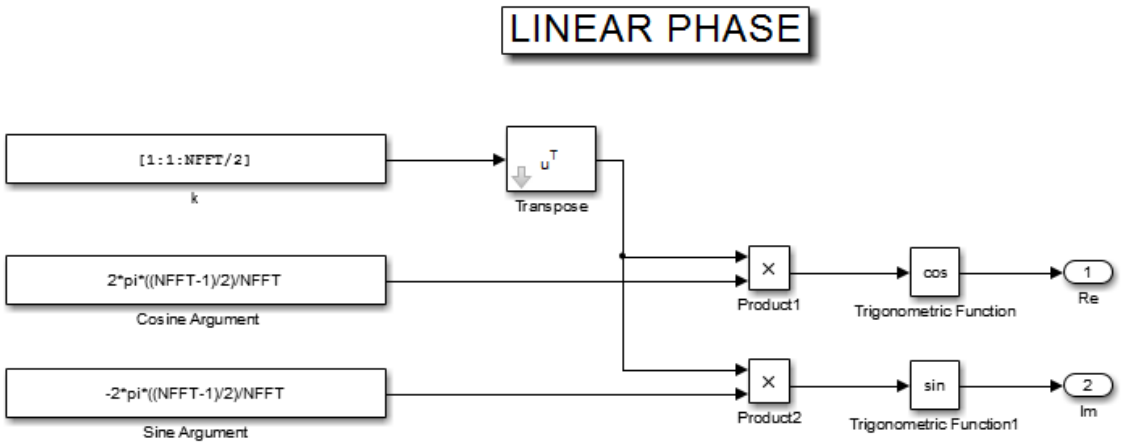

Figura 49: Subsistema Linear Phase para crear los vectores de longitud NFFT/2 con la parte real e imaginaria del filtro.

La respuesta en frecuencia del ecualizador de 30 bandas se muestra en la figura [50], junto con la fase perfectamente lineal para la configuración del ecualizador sin realce ni atenuación en todas sus bandas.

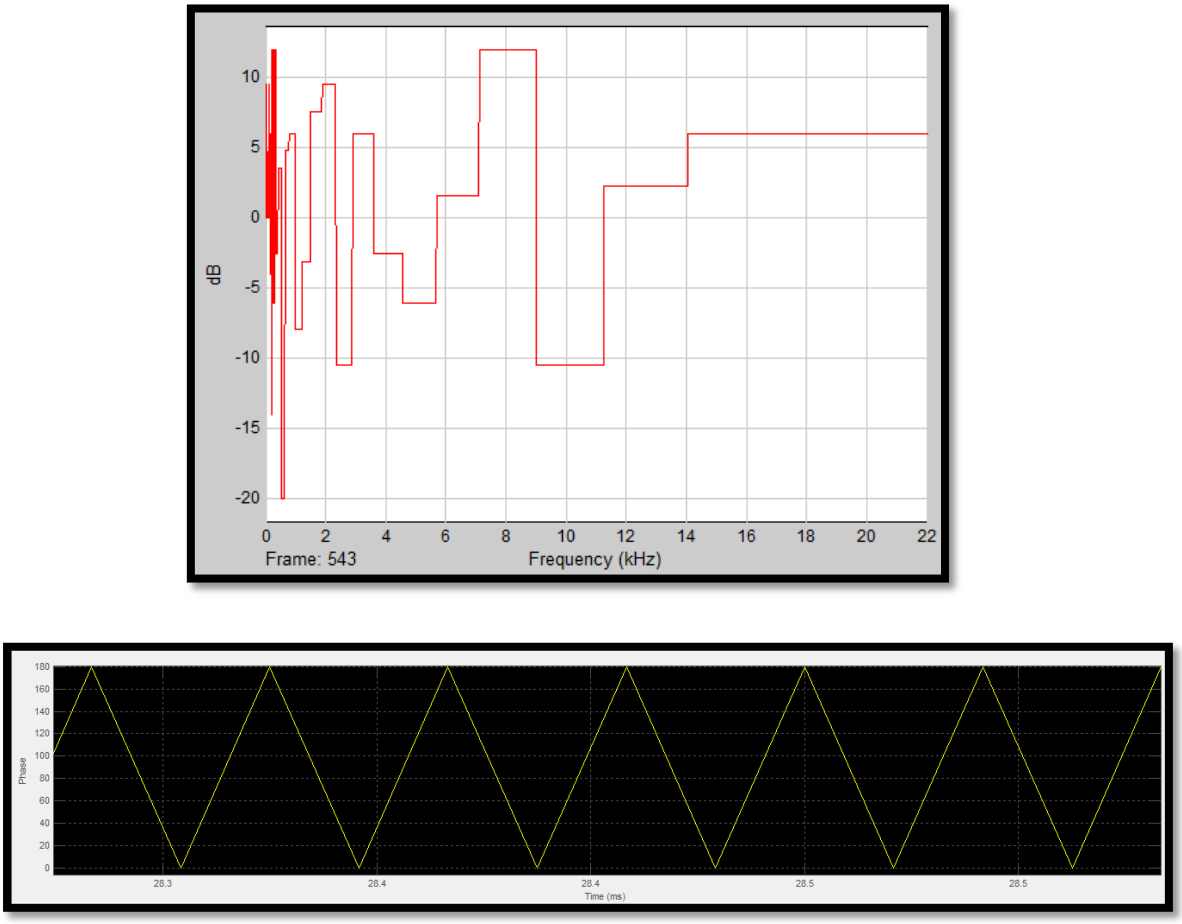

Figura 50: Respuesta en frecuencia del ecualizador de 30 bandas (arriba). Fase de la respuesta al impulso del ecualizador (abajo).

Finalmente, la figura [51] muestra las estructuras que realizan los procesos de análisis y síntesis. Las ventanas están configuradas con una longitud de wLen con un solape del 50%. La FFT realizada es de 2048 muestras, valor con el que se obtiene una buena estimación espectral, por lo que el tamaño de la ventana no debe ser muy grande con respecto a este valor para evitar un retardo de procesado elevado. Para evitar modificaciones en la amplitud tras los procesos de *FFT* e *IFFT*, en ambos bloques debe desactivarse la opción *„Scale result by FFT length"*. La construcción del bloque *Overlap-add* se explica con detalle en los apartados [7] y [6.3.].

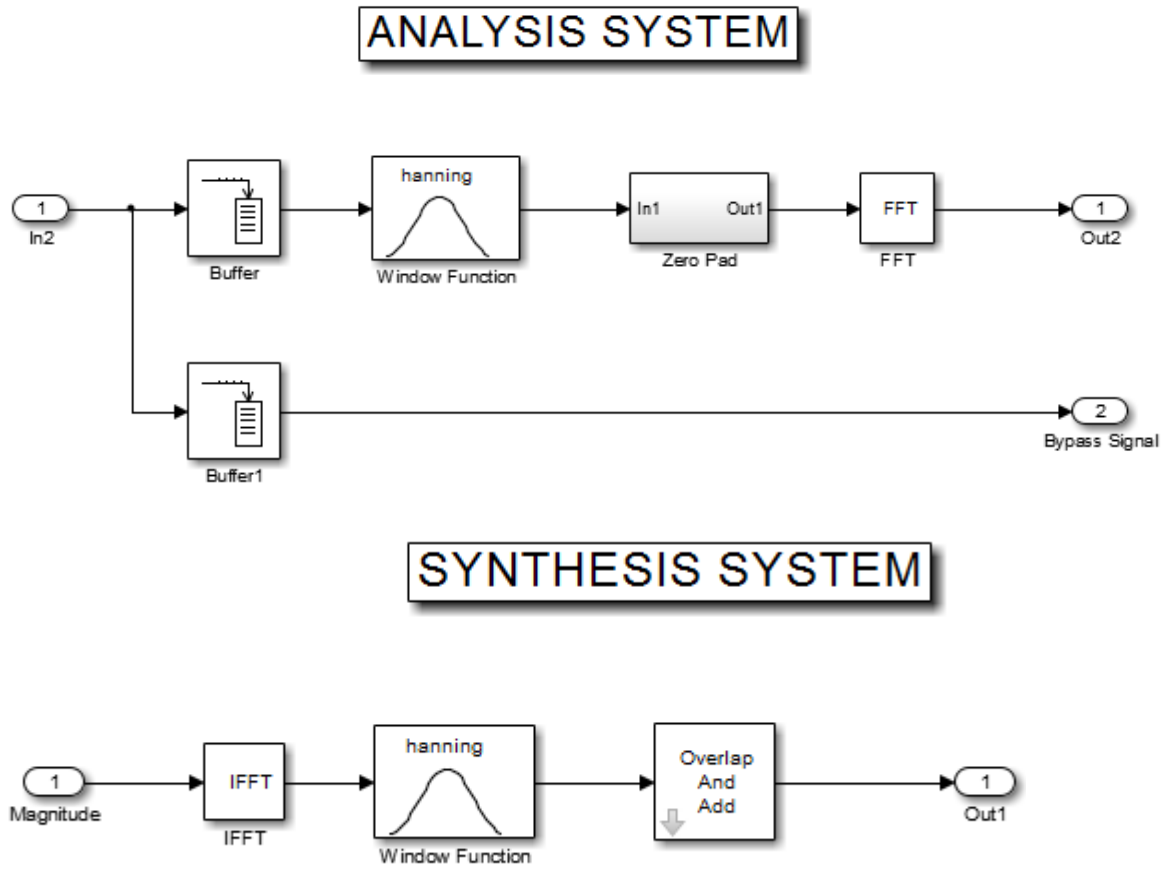

Figura 51: Subsistemas de análisis y síntesis del ecualizador gráfico FFT.

3.4. CONCLUSIONES

Es en este ámbito donde la biblioteca de *DSP System Toolbox* y Simulink destacan en cuanto al procesado digital de la señal. Esta biblioteca presenta una gran cantidad de métodos y bloques para realizar simulaciones y pruebas con todo tipo de filtros con una rapidez mucho mayor a la codificación en Matlab. Los filtros y ecualizadores pueden ser configurados muy rápidamente seleccionando únicamente los parámetros de su plantilla de especificaciones, y evitando al usuario introducirse en el

fondo matemático del diseño de filtros digitales. A pesar de estas posibilidades, en este proyecto se ha optado por una programación de los ecualizadores de más bajo nivel, de forma que se pudiera comprobar y explotar las posibilidades de Simulink en estos niveles.

Por tanto, los modelos desarrollados son un ejemplo de las distintas posibilidades que ofrece el programa para ser llevados a cabo, pudiendo optar por distintos caminos para llegar al mismo resultado. Así, sirve como ejemplo los bloques de *STFT* que realizan el mismo procedimiento que los bloques de análisis y síntesis diseñados en el algoritmo del ecualizador *FFT*. Otro claro ejemplo son los bloques de filtros que podrían sustituir perfectamente al diseño realizado por coeficientes de la función de transferencia de los ecualizadores tipo *Shelving* y *Peak.*

En cuanto a los algoritmos en sí, todavía hay mucho de qué hablar acerca de la ecualización en el mundo del audio. Empresas importantes como *RANE* con su ecualizador gráfico de Q perfecto [ref. 13] están dando pasos para acercarse cada vez más al ecualizador ideal, y la búsqueda de este algoritmo continúa. El objetivo es lograr un ecualizador cuya respuesta sea exactamente la deseada por el usuario, sin cambios en amplitud y sin interacción entre bandas. Se desea, por tanto, que la configuración establecida del ecualizador se corresponda al cien por cien con la ecualización realizada a la señal de audio.

Existen por otro lado otros ámbitos como son el filtrado y ecualización adaptativos, capaces de modificar en tiempo real y sin interacción del usuario la configuración de sus filtros y ecualizadores. Estos tienen un sinfín de aplicaciones, dentro y fuera del mundo del audio, y son sistemas funcionales aplicados actualmente y que pueden ser objeto de investigación y mejora.

Finalmente, el proceso de *STFT* es un método utilizado por una gran cantidad de algoritmos los cuales trabajan en tiempo real con señales de audio. A pesar de que este método se ha utilizado de forma básica en este algoritmo, se hace especial hincapié en su funcionamiento en el apartado [7] de *Vocoders*, donde su implementación es más compleja y requiere de un análisis más profundo.

4. PROCESADO DE RETARDOS

4.1. INTRODUCCIÓN

Las técnicas de procesado digital permiten conseguir efectos con retardos muy complejos y utilizados en el mundo del audio. La creciente potencia de los ordenadores y *DSPs* actuales permite la aplicación de ***delays*** de alto orden con un gran número de operaciones. Esto facilita la obtención de reverberadores de calidad con una gran densidad de ecos, los cuales se asemejan cada vez más a reverberaciones reales de recintos. El procesado con retardos permite también la obtención de efectos estéticos utilizados por músicos en sus instrumentos o por ingenieros en el proceso de mezcla para aportar un sonido más "brillante" al resultado final. Los efectos en esta línea más utilizados son:

- ***Chorus:*** Como su propio nombre indica, un *chorus* es un efecto que produce un eco muy cercano a la señal directa, dando la impresión de que la señal está siendo producida por dos o más fuentes iguales a la vez. Es muy utilizado para voces musicales. Se consigue mediante un retardo de entre 15 y 30 ms, modulado por una señal aleatoria filtrada en baja frecuencia.
- ***Flanger:*** Se trata de un efecto en el que se modula un retardo menor a 15ms con una frecuencia de oscilación muy baja de 1Hz. Este proceso produce un sonido metalizado oscilante sobre todo en alta frecuencia.
- ***Vibrato:*** El *vibrato* es un efecto causado al producir un retardo modulado en el tiempo por un oscilador de baja frecuencia. Este proceso produce un cambio constante de baja magnitud en el pitch de la señal, de forma que se produce una sensación de ida y vuelta del sonido. Es común el uso de un oscilador con una frecuencia entre 0.1 y 14 Hz, mientras que los retardos utilizados varían entre 1 y 10ms.
- ***Reverberador:*** El más común y el principal objeto del procesado de retardo es la reverberación. Antiguamente, el proceso de grabación de cualquier pieza musical exigía unas condiciones de reverberación determinadas del recinto en el que se realizara la grabación, quedando esta reverberación presente en la mezcla final. Hoy en día, las posibilidades de procesado digital son tan inmensas que la mejor forma de añadir las condiciones de reverberación deseadas es grabar la señal en ausencia de reverberación, lo más seca posible, de forma que se pueda añadir al final el efecto de *reverb* que se desee. Existen una gran multitud de algoritmos complejos distintos para lograr reverberaciones de distinto tipo y calidad, así como multitud de soluciones comerciales, tanto en formato de *plugin* como en equipo de rack.

Figura 52: Equipo reverb para rack REV100 del fabricante YAMAHA.

4.2. DISEÑO DE UN REVERBERADOR DE SCHROEDER

Uno de los primeros algoritmos para simular reverberaciones naturales de forma electrónica fue desarrollado por **Schroeder** en 1961 [ref. 7]. Schroeder propuso un sistema de filtros peine combinados en paralelo con filtros paso todo en serie, cada uno de los cuales presenta distintas ganancias y retardos. El esquema general de un reverberador de Schroeder se presenta en la figura [53]. Las unidades de la izquierda en paralelo corresponden a las estructuras de los filtros tipo peine. Las unidades de la derecha en serie son las estructuras correspondientes a filtros paso todo. Schroeder propuso un sistema formado por cuatro filtros peine en paralelo unidos con dos filtros paso todo en cascada.

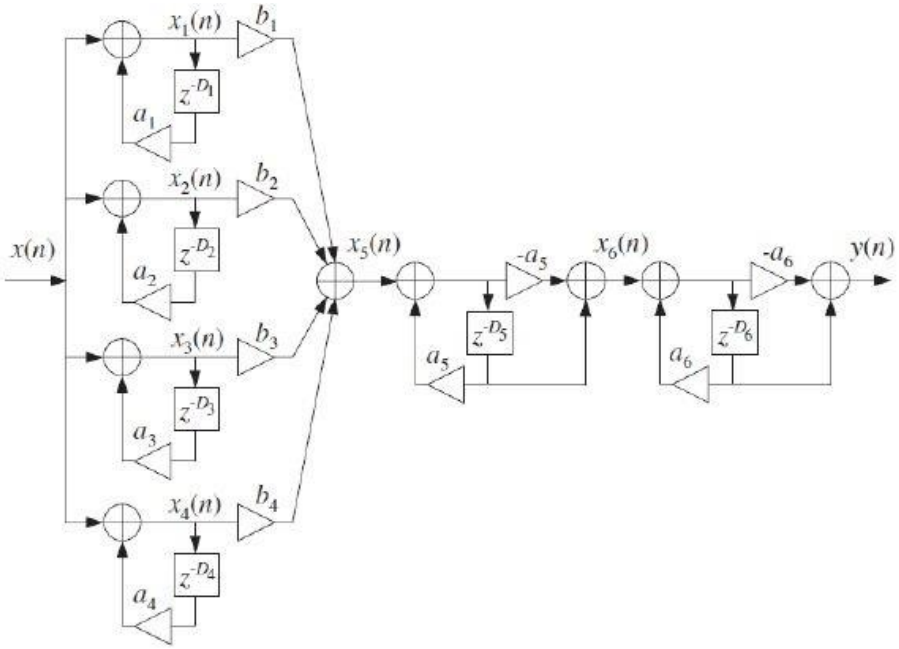

Figura 53: Esquema general de un reverberador de Schroeder.

Para poder entender el funcionamiento de un reverberador de estas características, es necesario explicar el funcionamiento de los filtros IIR tipo peine y los filtros IIR paso todo, es decir, cómo son sus estructuras y la respuesta en frecuencia que presentan, la cual debe producir una coloración de la señal lo más mínima posible para evitar sonidos indeseados.

4.2.1. FILTROS IIR TIPO PEINE:

En primer lugar, la generación de un eco con un retardo D tiene la siguiente expresión, siendo **D** el retardo en muestras $(D = \frac{\tau}{T_s})$ y **a** la ganancia de lazo:

$$y(n) = x(n) + ax(n - D) \quad (51)$$

Si esta expresión es generalizada para infinitos ecos, la ecuación en diferencias del sistema cumple con la expresión [52]:

$$y(n) = x(n) + ax(n - D) + a^2x(n - 2D) + a^3x(n - 3D) + \cdots \quad (52)$$

Calculando la respuesta al impulso:

$$h(n) = \delta(n) + a\delta(n - D) + a^2\delta(n - 2D) + a^3\delta(n - 3D) + \cdots \quad (53)$$

La transformada Z de esta respuesta cumple con la expresión siguiente:

$$H(z) = 1 + az^{-D} + a^2z^{-2D} + a^3z^{-3D} + \cdots = \frac{1}{1 - az^{-D}} = \frac{Y(z)}{X(z)} \quad (54)$$

Finalmente, calculando la transformada inversa, el algoritmo del filtro es:

$$y(n) = x(n) + ay(n - D) \quad (55)$$

Este sistema es un reverberador simple con una respuesta en frecuencia denominada tipo peine o comb filter, debido a su similitud con este objeto. La respuesta presenta una serie de picos equiespaciados en las posiciones 2kπ/D, con k=1,2,3…D-1. El rizado que presente entre los máximos y mínimos se corresponde con la expresión [56]. Este filtro se lleva a cabo mediante la estructura recursiva mostrada en la figura [54].

$$Rizado(dB) = 20log\frac{1 + a}{1 - a} \quad (56)$$

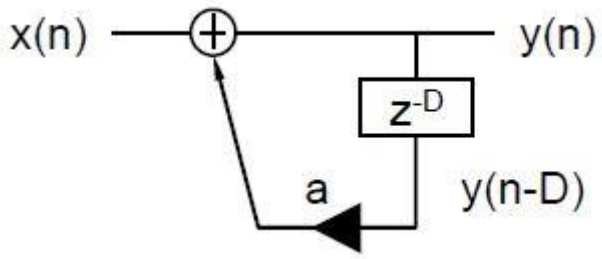

Figura 54: Algoritmo recursivo de un filtro IIR tipo peine.

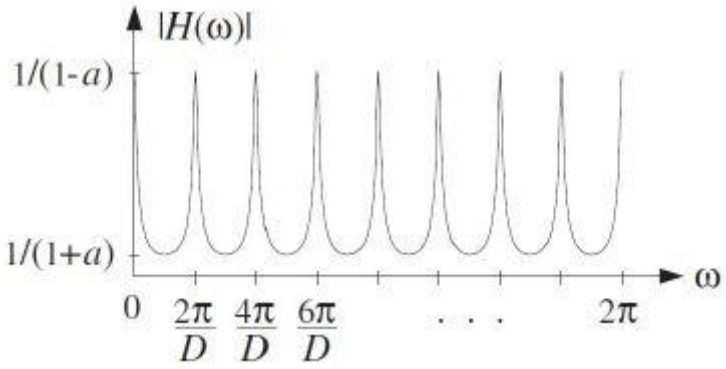

Figura 55: Respuesta en frecuencia de un filtro IIR tipo peine.

Una forma rápida de probar este algoritmo es identificando coeficientes y representando la respuesta en frecuencia mediante el comando *fvtool(b,a)* en Matlab. Según la expresión [57]:

$$H(z) = \frac{1}{1 - az^{-D}} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_D z^{-D}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_D z^{-D}} \quad (57)$$

Identificando coeficientes:

$$b = [1,0,0,0, \ldots, 0] \quad (58)$$

$$a = [1,0,0,0, \ldots, -a] \quad (59)$$

Se construyen estos vectores en Matlab para un retardo de D=32 y una ganancia de lazo a=0.9, obteniendo la respuesta en frecuencia, la respuesta al impulso, y el diagrama de polos y ceros. Puede observarse que la respuesta en frecuencia presenta D máximos (32 en este caso). La respuesta al impulso presenta una forma cercana a la que se desea lograr con el reverberador, aunque con poca densidad de ecos. El diagrama de polos y ceros muestra una serie de D polos espaciados 2π/D, estando todos los ceros en el origen.

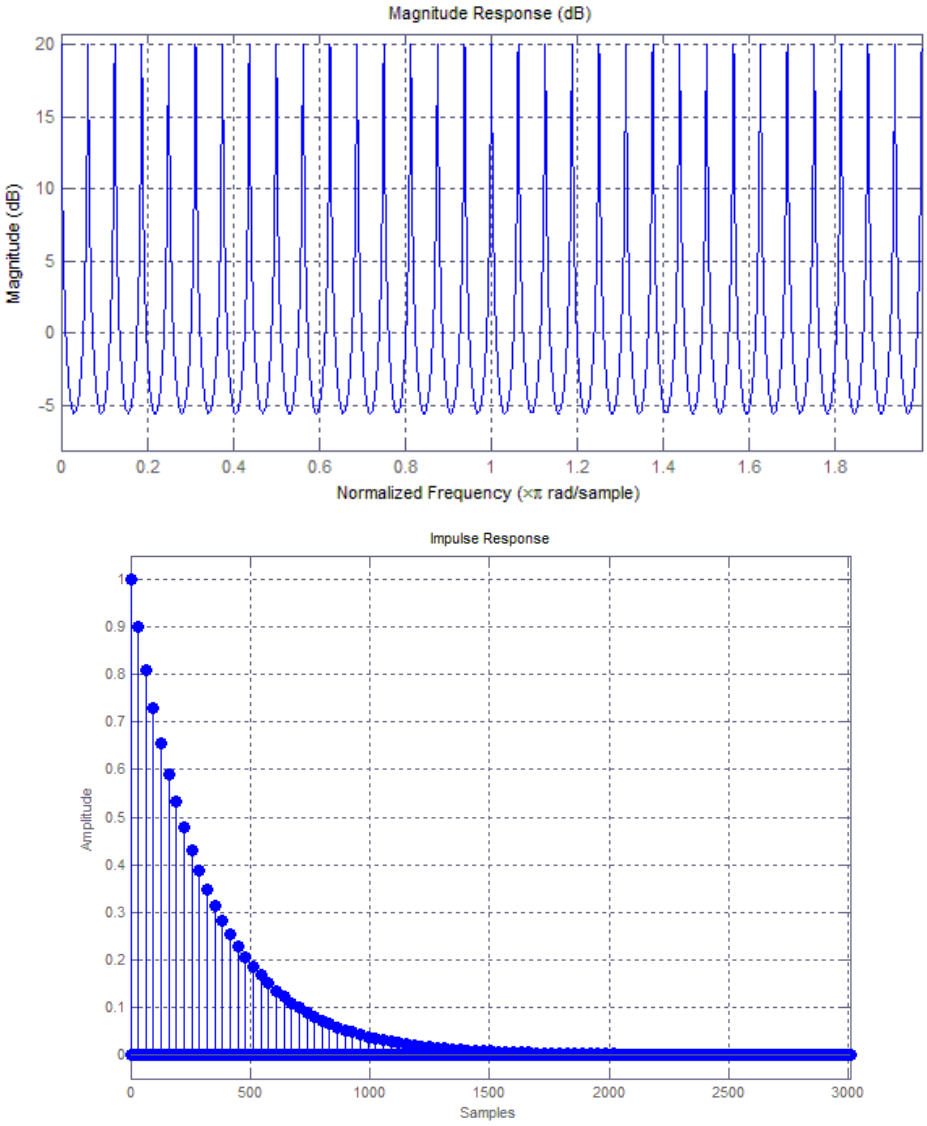

Figura 56: Respuesta en frecuencia (arriba) y respuesta al impulso (abajo) de un filtro IIR tipo peine con D=32 y a=0.9.

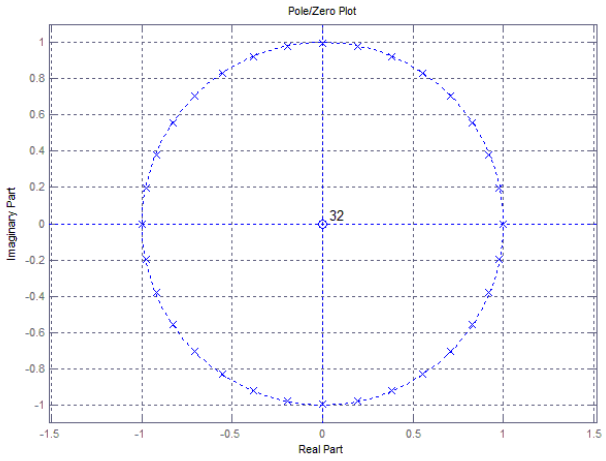

Figura 57: Diagrama de polos y ceros de un filtro IIR tipo peine con D=32 y a=0.9.

La respuesta de este filtro produce realces significativos en las frecuencias donde están situados los polos, dando lugar a un sonido metálico molesto y no deseado. Una forma de evitar este efecto es introducir un filtro paso bajo de orden 1 en el lazo de realimentación, tal y como muestra la figura [58]. En una respuesta real, las altas frecuencias se atenúan más rápidamente por las propiedades de los materiales de la sala donde se producen las reflexiones (ecos) del sonido. La función del filtro paso bajo es simular también esta disminución del nivel de las altas frecuencias.

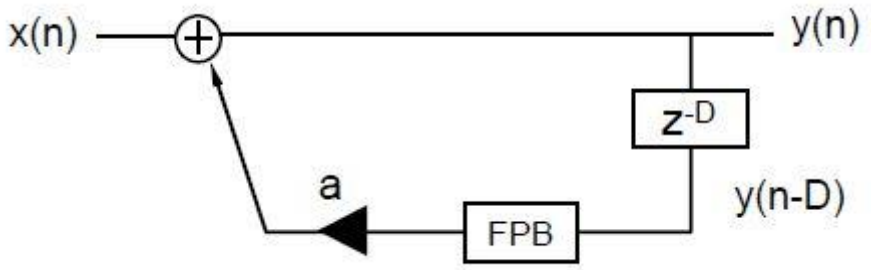

Figura 58: Algoritmo recursivo de un filtro IIR tipo peine con filtro paso bajo en el lazo.

4.2.2. FILTROS IIR PASO TODO:

Como su propio nombre indica, estos filtros presentan una respuesta en frecuencia plana, por lo que no producen coloración de la señal. La respuesta en frecuencias de los filtros anteriores realza en gran medida las frecuencias a las que se encuentran los picos, produciendo una coloración excesiva, por lo que estos filtros actúan disminuyendo este efecto a la vez que permiten añadir una gran densidad de ecos a la señal de entrada al ser colocados en serie. La ecuación en diferencias de este tipo de filtros cumple con la expresión [60]:

$$y(n) = -ax(n) + x(n - D) + ay(n - D) \quad (60)$$

Calculando la respuesta en frecuencia:

$$Y(z) = -aX(z) + z^{-D}X(z) + az^{-D}Y(z) \quad (61)$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{-a + z^{-D}}{1 - az^{-D}} \quad (62)$$

La estructura que cumple con la ecuación en diferencias anteriormente descrita se muestra en su forma canónica en la figura [59]. Se muestra también en esta figura la estructura en su forma paralela propuesta por Schroeder.

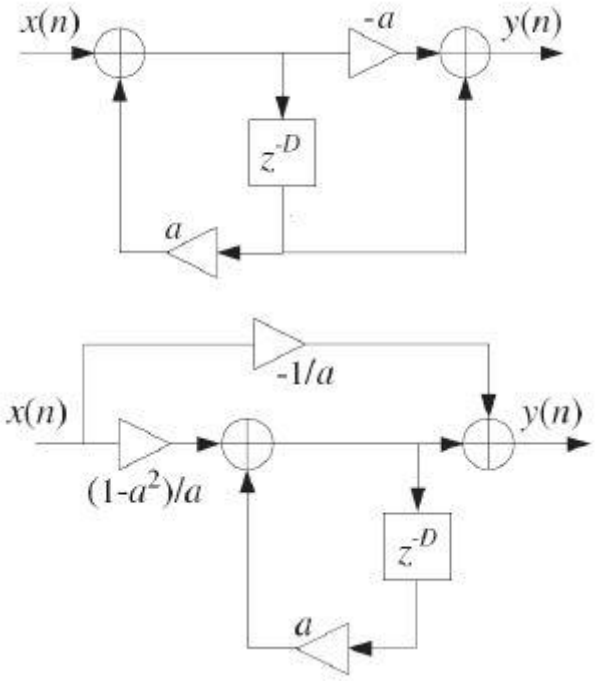

Figura 59: Estructura canónica (arriba) y estructura en paralelo (abajo) de un filtro IIR paso todo.

De la misma manera, identificando coeficientes de la transformada Z se representa la respuesta en frecuencia de un filtro paso todo, así como la respuesta al impulso y el diagrama de polos y ceros mediante la función de Matlab *fvtool*.

$$H(z) = \frac{-a + z^{-D}}{1 - az^{-D}} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_D z^{-D}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_D z^{-D}} \quad (63)$$

Identificando coeficientes:

$$b = [-a, 0,0,0, \ldots, 1] \quad (64)$$

$$a = [1,0,0,0, \ldots, -a] \quad (65)$$

Se construyen estos vectores en Matlab para un valor de D =32 y a=0.9. La respuesta en frecuencia, como se puede comprobar, es una respuesta totalmente plana. La respuesta al impulso presenta una caída exponencial con una forma cercana a la reverberación que se desea conseguir.

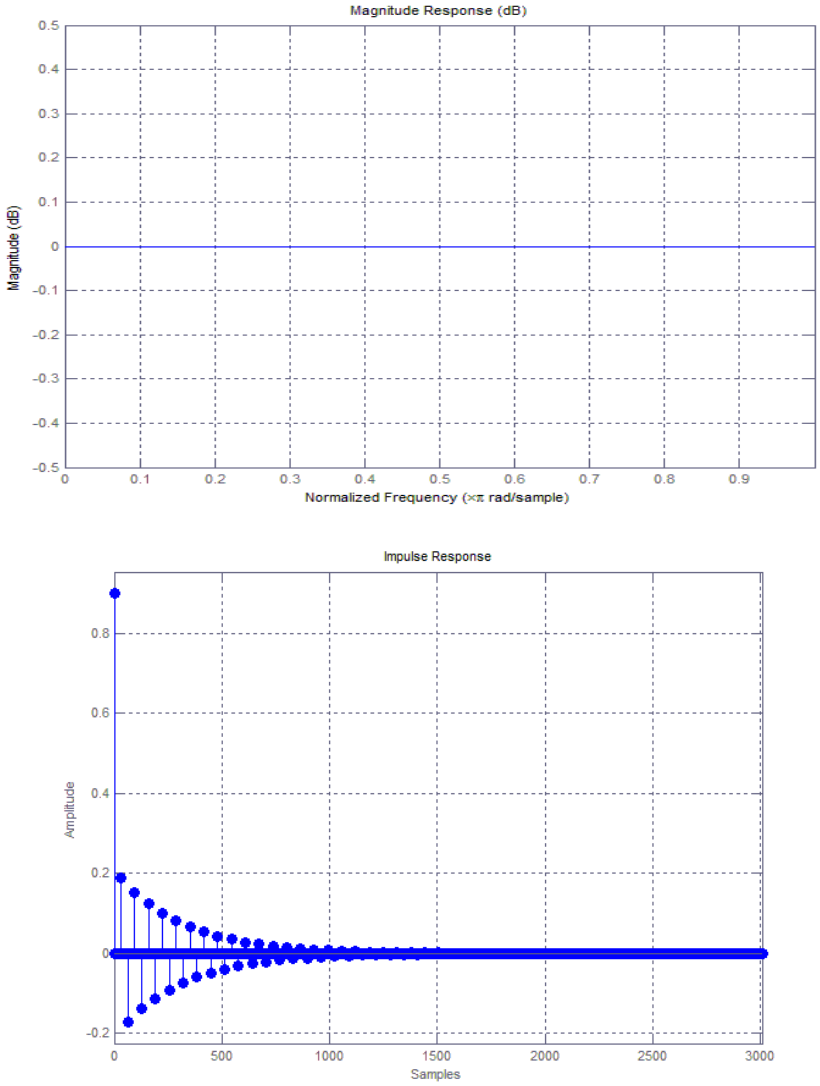

Figura 60: Respuesta en frecuencia (arriba) y respuesta al impulso (abajo) de un filtro IIR paso todo con D=32 y a =0.9.

Finalmente, el diagrama de polos y ceros presentan 32 polos y ceros espaciados 2π/D. Estos ceros y polos no están en la misma posición, si no que los ceros se encuentran más alejados que los polos, tal y como se muestra en la figura [61]. Los polos están situados en un radio de $\sqrt[D]{a}$ y los ceros en un radio de $\sqrt[D]{\frac{1}{a}}$.

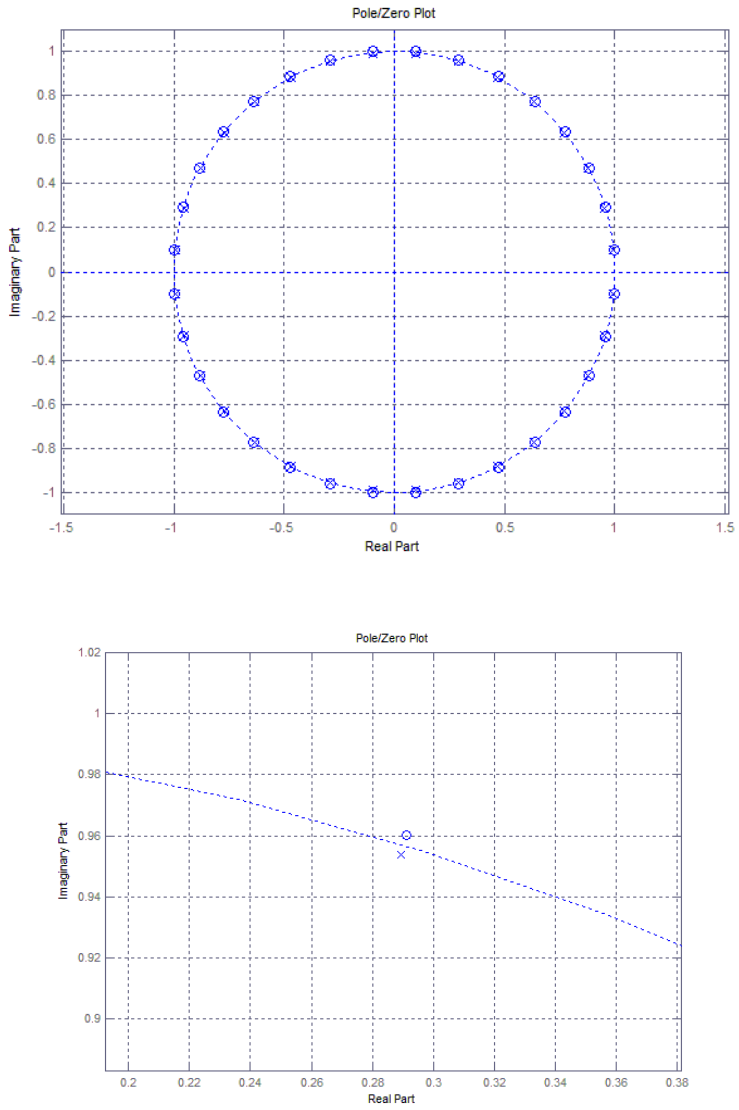

Figura 61: Diagrama de polos y ceros de un filtro IIR paso todo con D=32 y a=0.9.

4.2.3. DISEÑO DEL REVERBERADOR EN SIMULINK

Una vez establecidas las bases de los filtros a utilizar y la estructura del reverberador de Schroeder, la construcción en Simulink se lleva a cabo mediante los bloques disponibles en la biblioteca de *DSP System Toolbox*, los cuales permiten la construcción de las estructuras descritas anteriormente. Por tanto, el diseño consta de 6 subsistemas, cada uno de los cuales presenta en el nivel inferior la estructura de los filtros IIR peine y paso todo. El control del reverberador se realiza seleccionando el **tiempo de reverberación** deseado. El tiempo de reverberación, según Sabine, se define como el tiempo que tarda en caer la señal 60 dB desde que la emisión es interrumpida dentro de un recinto. Además, se incluyen los controles de ***Dry*** y ***Wet***, que permiten controlar la cantidad de señal directa y señal reverberante que se desea. Un control de *Bypass* actúa como conmutador entre la señal sin procesar y la señal procesada.

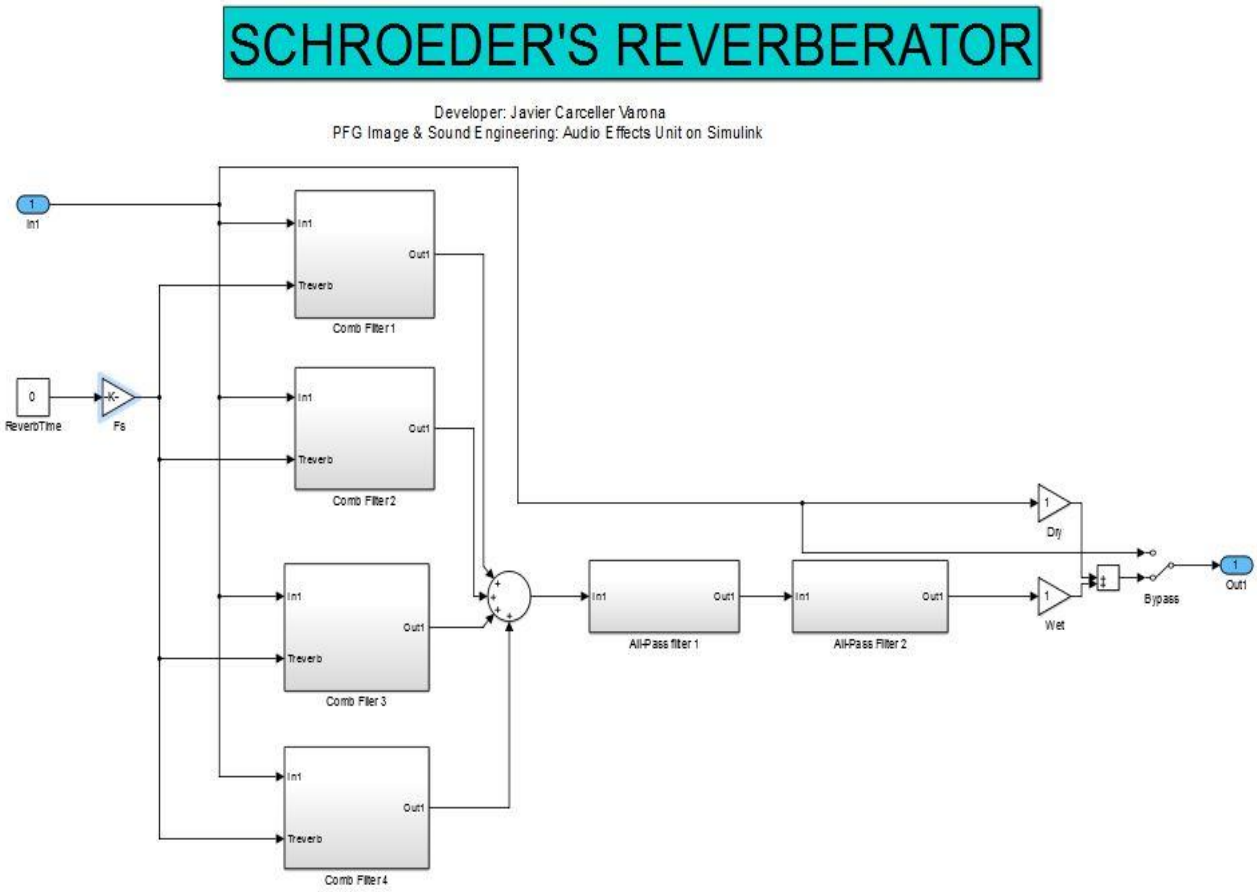

Figura 62: Estructura general de un reverberador de Schroeder en Simulink.

Para que los filtros peine dependan del tiempo de reverberación, debe encontrarse la relación entre la ganancia del lazo y el tiempo de reverberación T establecido. En cada iteración del lazo, el nivel de la señal cae -20log(g) dB, siendo τ(s) el retardo producido por la iteración. Cumpliendo con la definición anterior del tiempo de reverberación:

$$Tr = \frac{60}{-20log(g)}\tau \tag{66}$$

Siendo $D = \frac{\tau}{T_s}$:

$$Tr = \frac{3}{log(\frac{1}{g})}DT_s = \frac{3}{log(\frac{1}{g})}\frac{D}{f_s} \tag{67}$$

Despejando la ganancia de lazo:

$$g = 10^{\frac{-3D}{Trf_s}} \tag{68}$$

Por lo tanto, la ganancia de lazo es directamente proporcional al tiempo de reverberación. A mayor tiempo de reverberación, la ganancia de lazo tendrá un valor mayor, mientras que a menor tiempo de reverberación, la ganancia de lazo tendrá un valor menor.

Una vez establecida esta relación, la estructura descrita de un filtro IIR tipo peine se construye en Simulink siguiendo la figura [65]. Como entradas se encuentran la señal directa y el tiempo de reverberación multiplicado por la frecuencia de muestreo previamente. Como bloque para realizar el retardo se escoge el bloque *Variable Integer Delay*, el cual permite seleccionar el retardo deseado mediante una variable de entrada. Es importante marcar la casilla *Disable direct feedthrough* en caso de existir un lazo de realimentación para evitar que se produzca un bucle algebraico infinito. Posterior al retardo se encuentra el filtro paso bajo de primer orden para reducir los efectos de coloración del lazo no deseados. Este filtro cumple con la función de transferencia descrita en la figura [65]. El bloque de ganancia utiliza un bloque de código en Matlab para calcular la ganancia en función del tiempo de reverberación. Este código se muestra a continuación:

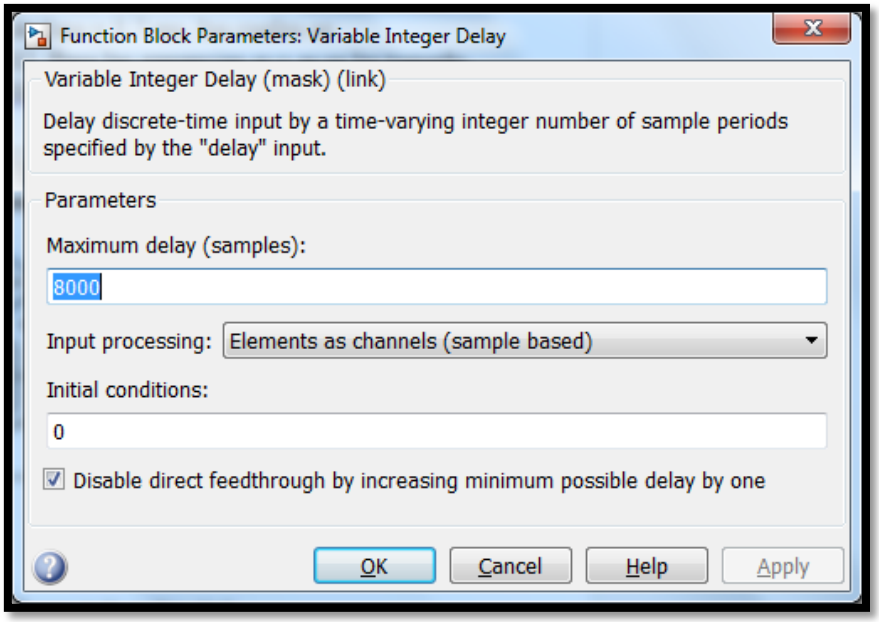

Figura 63: Ventana de configuración del bloque Variable Integer Delay.

```
function [y,g] = VariableGain(x,D,Tr)
%This function calculates the gain of the loop with the Reverb Time.
%x: Direct signal
%D: Delay of the loop
%Tr: Reverb Time* fs(sample frequency)
g= 10^(-3*D/Tr);
y=g*x;
```

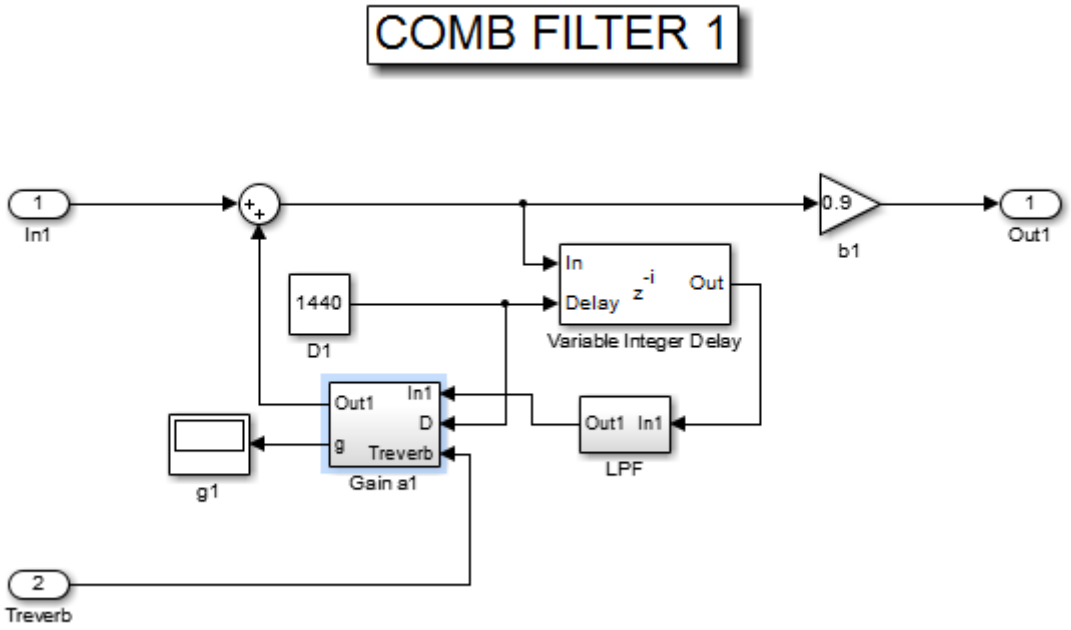

Figura 64: Estructura de un filtro IIR tipo peine en Simulink.

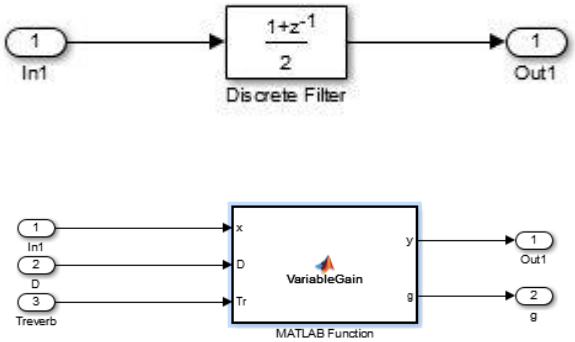

Figura 65: Filtro paso bajo del lazo (izquierda) y bloque calculador de la ganancia de lazo (derecha).

Para conseguir un efecto de reverberación natural, Schroeder propuso un conjunto de retardos para cada uno de los filtros peine, en un rango de valores entre 30ms y 45 ms. Estos valores se corresponden con retardos cuantizados en muestras de D_1=1440 y D_4=2160, teniendo en cuenta una frecuencia de muestreo de 48kHz. Los valores tomados

deben estar dentro de un rango de 1:1.5 entre el primer y el último retardo. Además, las ganancias de salida de cada filtro denominadas como las variables b_n deben ser menores según el retardo sea mayor, para conseguir una mayor similitud con la forma de onda de una reverberación. Los valores para estas variables escogidos en el reverberador se muestran en la tabla [2].

Tabla 2: Valores de retardo y ganancia de salida de los filtros IIR tipo peine y paso todo.

	D_n	b_n
Comb Filter 1	1440	0.9
Comb Filter 2	1656	0.8
Comb Filter 3	1872	0.7
Comb Filter 4	2160	0.6

Los filtros IIR paso todo se han construido siguiendo la estructura canónica descrita anteriormente. Schroeder propuso para estos dos filtros en serie retardos de 5ms y 1.7 ms, los cuales se corresponden con valores en muestras de D_5= 240 y D_6 = 82, teniendo en cuenta una frecuencia de muestreo de 48kHz. Para las ganancias a_5 y a_6 se ha tomado el valor de 0.7. La figura [66] muestra la estructura de los filtros paso todo en Simulink.

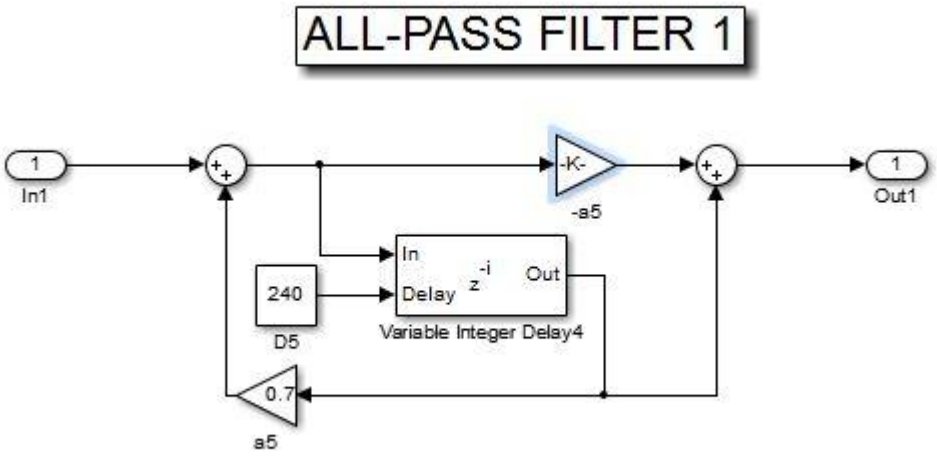

Figura 66: Estructura de un filtro IIR paso todo en Simulink.

4.3. RETARDOS MODULADOS EN EL TIEMPO

La posibilidad de producir retardos variables en el tiempo modulados por distintos tipos de señales permite el diseño de efectos de audio con distintas aplicaciones, en su mayor parte estéticas. Como se han descrito previamente, los más utilizados son el ***flanger***, el ***vibrato***, y el ***chorus***. Todos ellos utilizan retardos modulados. Sin embargo, estos retardos conllevan un problema. La modulación produce valores de retardos no enteros, los cuales no son aplicables en procesado de la señal discreta. La expresión [69] muestra la salida de un sistema con un retardo no entero.

$$y(n) = ax(n - (M + frac)) \tag{69}$$

La solución más utilizada a este problema es realizar una interpolación para calcular cada muestra de salida y(n), entre dos muestras adyacentes M y M+1. Existen distintos tipos de interpolaciones a utilizar en función de la aplicación. En este caso se ha escogido la **interpolación lineal**, cumpliendo con la expresión [70].

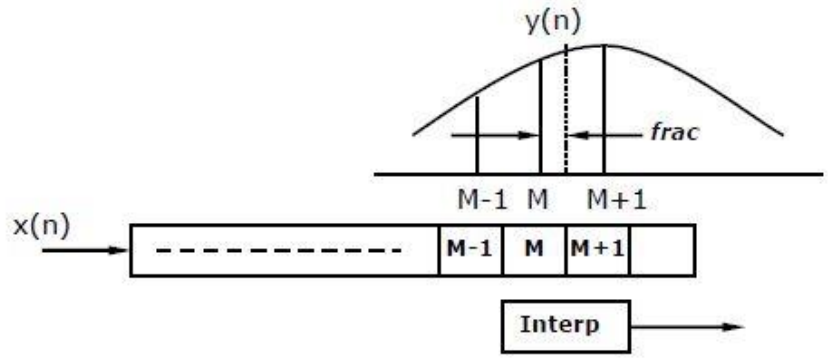

Figura 67: Interpolación lineal entre dos muestras adyacentes M y M+1.

$$y(n) = a[(1 - frac)x(n - M) + (frac)x\big(n - (M + 1)\big)] \tag{70}$$

Para el desarrollo de los efectos descritos anteriormente, existen al igual que con los reverberadores múltiples algoritmos distintos con distintos efectos sonoros y calidades. **Dattorro** [ref. 5] propuso una estructura universal modificando un filtro paso todo junto con un filtro tipo peine. Esta estructura, denominada **Estructura paso todo generalizada**, se muestra en la figura [68]. Dependiendo de los valores de ganancia y retardo escogidos la estructura actúa para implementar un tipo de efecto u otro.

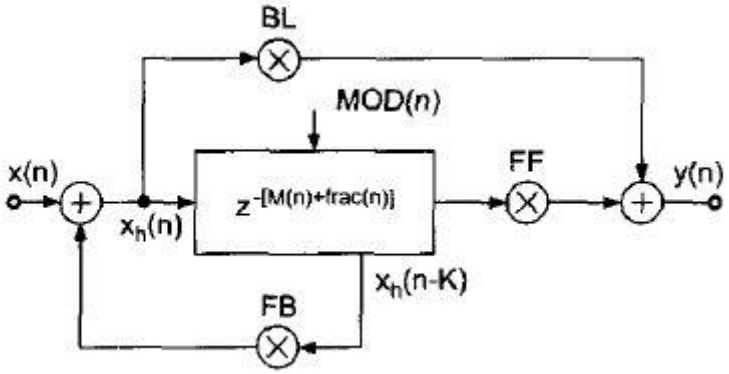

Figura 68: Estructura de un filtro paso todo generalizada para varios tipos de efectos.

Tabla 3: Valores de ganancia y retardos para la estructura generalizada de un filtro paso todo.

	BL(Blend)	FF(Feedforward)	Fb(Feedback)	Delay	Depth	Mod
Vibrato	0	1	0	0ms	0-3ms	0.1-5Hz sine
Flanger	0.7	0.7	0.7	0ms	0-2ms	0.1-1Hz sine
Chorus	0.7	1	-0.7	1-30ms	1-30ms	Lowpass Noise
Doubling	0.7	0.7	0	10-100ms	1-100ms	Lowpass Noise

4.3.1. DISEÑO DE UN ECHO DELAY EN SIMULINK

Partiendo de la estructura anterior, el efecto más sencillo a implementar es un *Echo Delay* sencillo. Este efecto consiste en añadir un único eco a la señal directa de forma constante. Musicalmente hablando, dependiendo de la figura de la nota que se desee doblar se elegirá el retardo a implementar. Como ejemplo, para un *tempo* de una pieza musical de 100bpms (negras por minuto), el tiempo de una figura de negra sería 60/100=600ms. Si se desease implementar un eco a ritmo de corchea para doblar el efecto de esta nota, sería necesario un retardo de 300ms. Cuanto el retardo es inferior a 100ms, un efecto de eco se convierte en un efecto denominado ***Doubling***, donde los ecos percibidos están muy cercanos a la señal directa, pero aún son diferenciados de esta.

A la hora de implementar el *Echo Delay* en Simulink, se ha establecido la estructura de un filtro paso todo generalizada para el efecto de *Doubling*, sin modulación. Se elimina por tanto el lazo de realimentación o *Feedback*, ya que se requiere únicamente un eco. La estructura en Simulink se muestra en la figura [69].

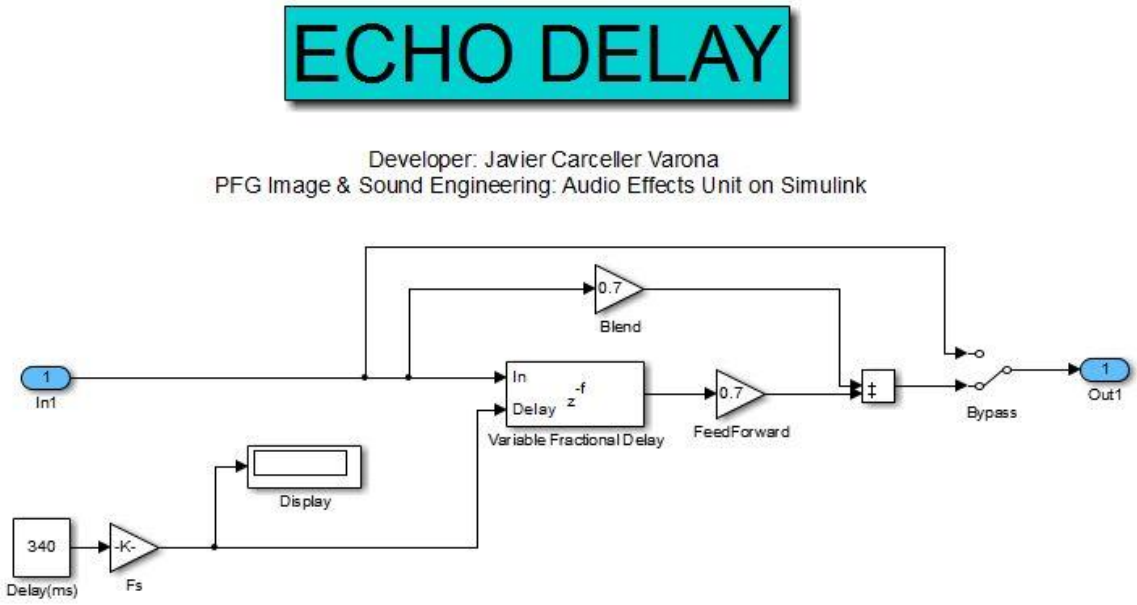

Figura 69: Estructura de un Echo Delay en Simulink.

Al contrario que en el reverberador, se ha utlizado el bloque *Variable Fractional Delay* para realizar el retardo de la estructura. Este bloque permite como entrada retardos fraccionarios, haciendo posible en su configuración la selección del algoritmo de interpolación. En este caso se selecciona la opción Interpolación Lineal, tal y como se muestra en la figura [70]. No es necesario marcar la casilla de *Disable Direct Feedthrough* ya que no existe lazo de realimentación.

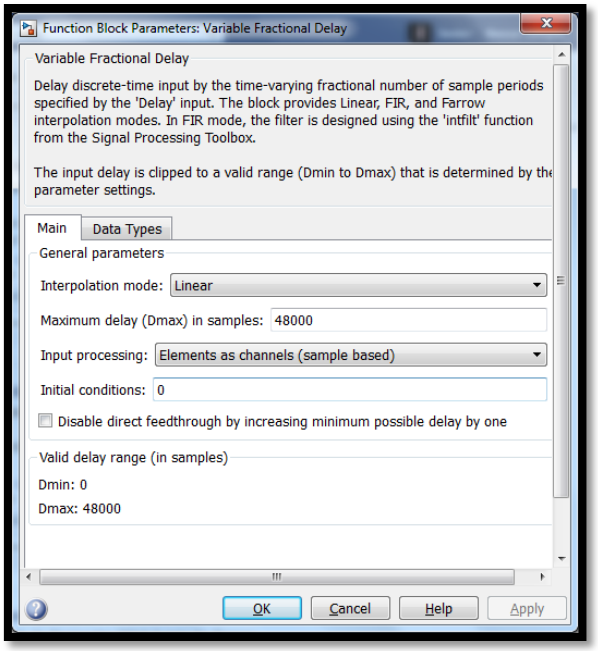

Figura 70: Ventana de configuración del bloque Variable Fractional Delay.

4.3.2. DISEÑO DE UN CHORUS EN SIMULINK

El efecto de *Chorus* implementado cumple de nuevo con la estructura generalizada de un filtro paso todo, manteniendo esta vez el lazo realimentado. El retardo para un *Chorus*, tal y como se ha explicado previamente, se mantiene dentro del rango de 1ms a 30ms. Este retardo es modulado por una señal de ruido aleatorio de baja frecuencia. La frecuencia debe ser suficientemente baja para evitar que el retardo cambie muy rápidamente, lo cual produce ruidos audibles indeseados en la señal. Al igual que en la estructura de los filtros IIR tipo peine, se incluye un filtro paso bajo de primer orden idéntico al de la figura [X] para reducir el sonido metálico producido por el filtro. El bloque de *Variable Fractional Delay* está configurado de la misma forma que en el caso del *Echo Delay*. La estructura en Simulink se muestra en la figura [71].

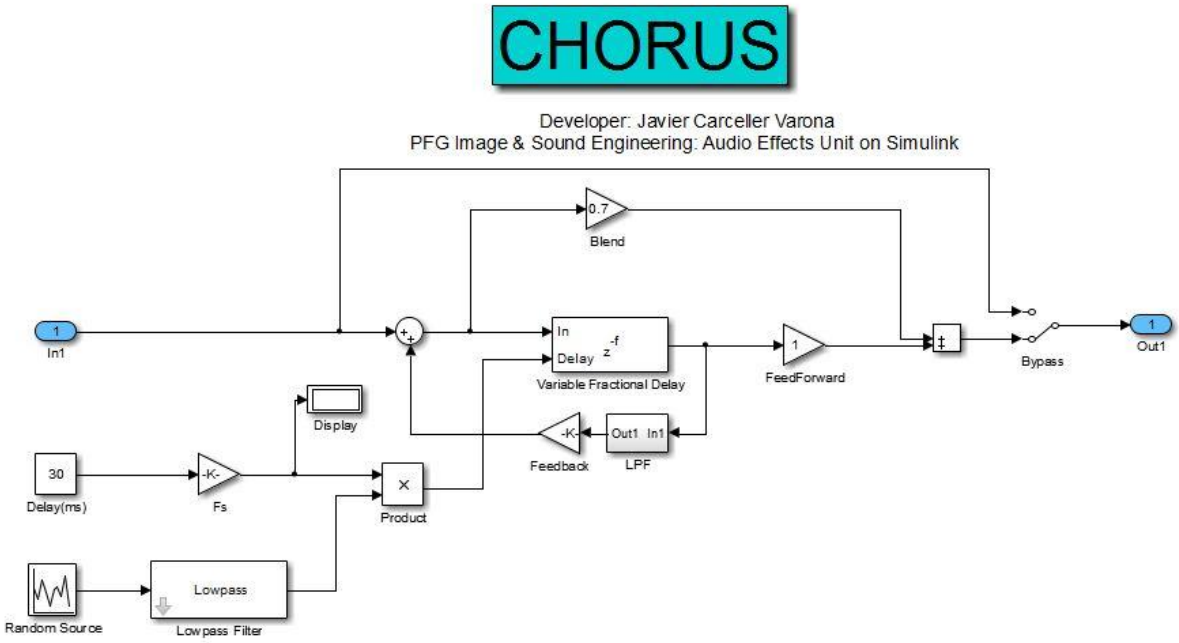

Figura 71: Estructura en Simulink de un Chorus mediante un filtro paso todo generalizado.

La parte moduladora está formada por un generador de señal aleatoria seguido de un filtro paso bajo que multiplica al retardo. En este caso, se ha utilizado el bloque de *Low pass Filter*, que permite realizar el filtrado de la señal de entrada estableciendo en su configuración los parámetros del filtro. La configuración utilizada se muestra en la figura [72], con una frecuencia de corte lo suficientemente baja para no producir efectos indeseados.

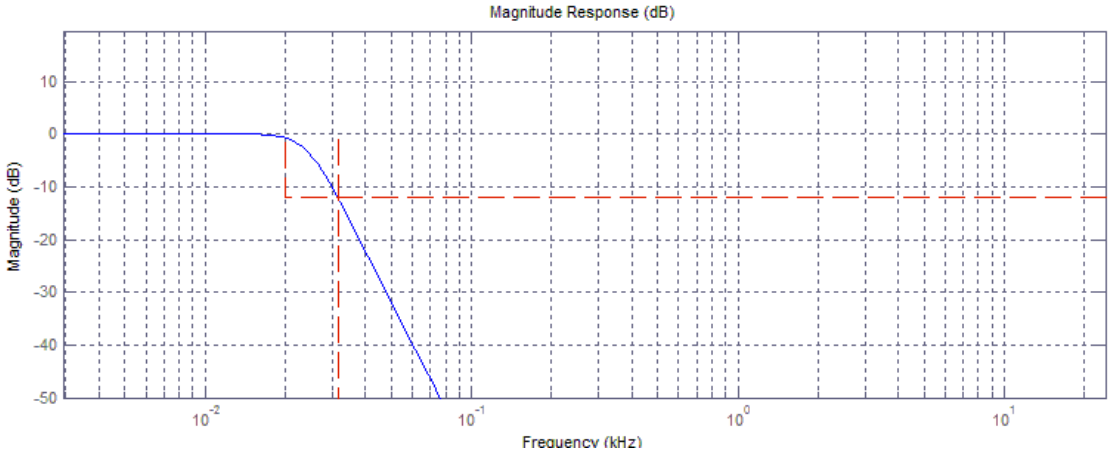

Figura 72: Respuesta en frecuencia del filtro paso bajo en el efecto de Chorus.

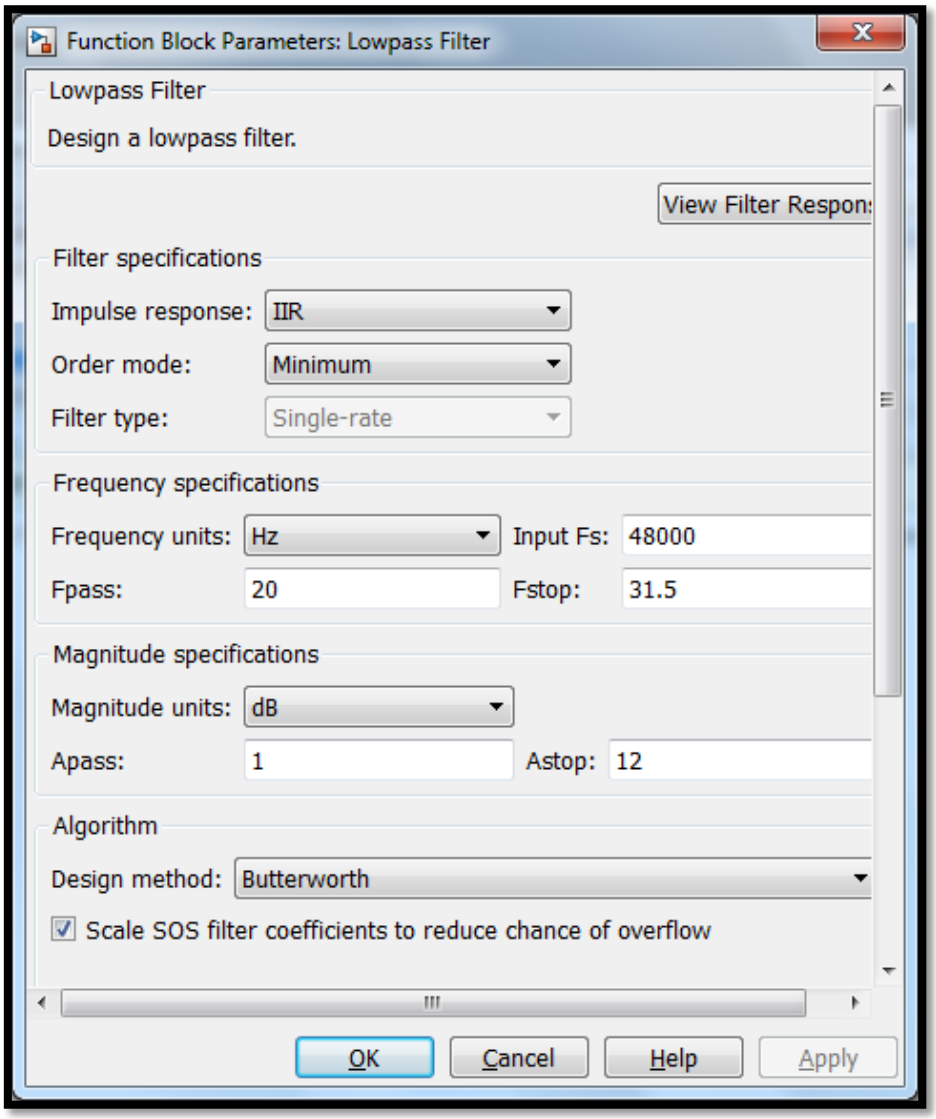

Figura 73: Configuración del bloque Low Pass Filter.

4.3.3. DISEÑO DE UN FLANGER EN SIMULINK

El efecto de un *Flanger* sencillo se consigue realizando un retardo menor a 15ms modulado por una señal sinusoidal de muy baja frecuencia. Este retardo se implementa en la estructura de filtro paso todo generalizada descrita anteriormente. Al igual que en el *Chorus*, se añade un filtro paso bajo de primer orden en el lazo de realimentación para atenuar el realce producido por el filtro en altas frecuencias. El retardo modulado cumple con la expresión [71], siendo la estructura del efecto en Simulink la mostrada en la figura [75]. D es el retardo a modular en muestras y F_d la frecuencia de la señal sinusoidal moduladora. Esta expresión entrega una señal d(n) sinusoidal con valores entre 0 y D.

$$d(n) = \frac{D}{2}[1 - cos(2\pi F_d n)] \quad (71)$$

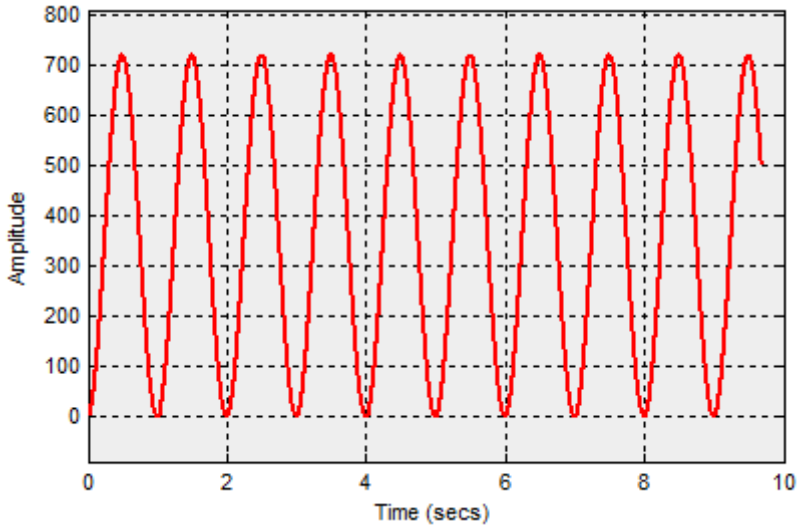

Figura 74: Señal de retardo modulado d(n) para D=720 muestras (τ=15ms) y F_d=1Hz.

El resultado para una señal de entrada sinusoidal de 10 Hz se muestra en la figura [76]. La señal de color rosa se corresponde con la salida del sistema. Puede visualizarse que la sinusoide se desplaza en frecuencia continuamente con respecto a la señal sin procesar, variando también en amplitud. Este efecto se incrementa en función del retardo D implementado.

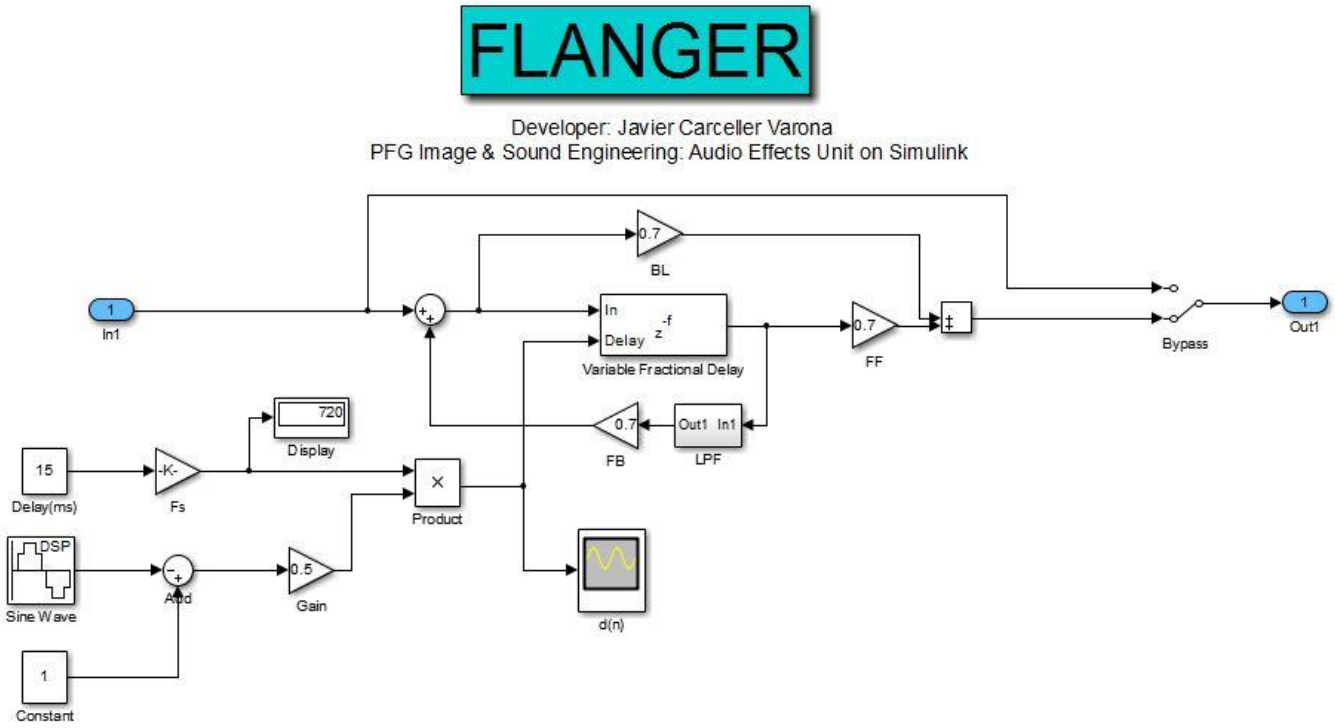

Figura 75: Estructura de un Flanger en Simulink mediante un filtro paso todo generalizado.

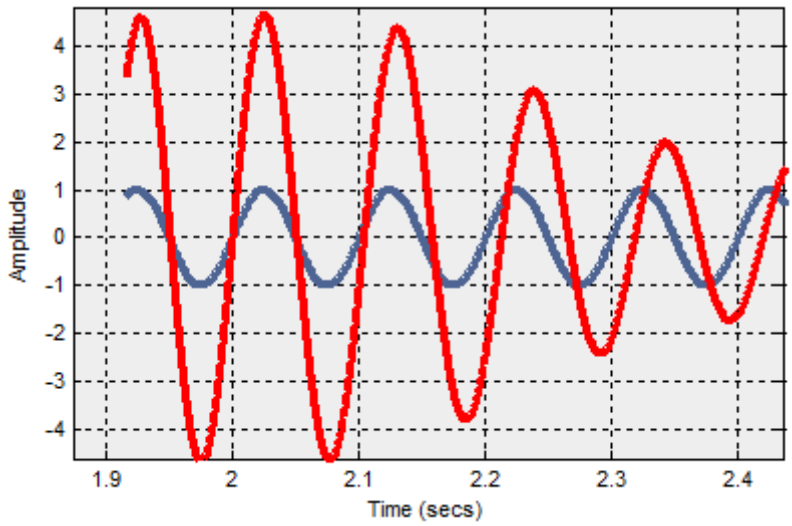

Figura 76: Effecto de Flanging para una señal de entrada sinusoidal de frecuencia 10 Hz. Frecuencia de la señal moduladora 1Hz. Retardo D =720 muestras.

4.3.4. DISEÑO DE UN VIBRATO EN SIMULINK

El efecto de *Vibrato* presenta un funcionamiento muy similar al de *Flanger*, con la diferencia de que el retardo D aplicado debe ser lo menor posible, en un rango entre 1ms a 10ms. Este retardo se modula de la misma manera con una señal sinusoidal con valores de amplitud entre 0 y 1, y una frecuencia en el rango de 0.1Hz a 14Hz. El *Vibrato* no requiere de lazo de realimentación, por lo que BL y FB se establecen con valor 0. La figura [77] muestra la estructura de un *Vibrato* en Simulink.

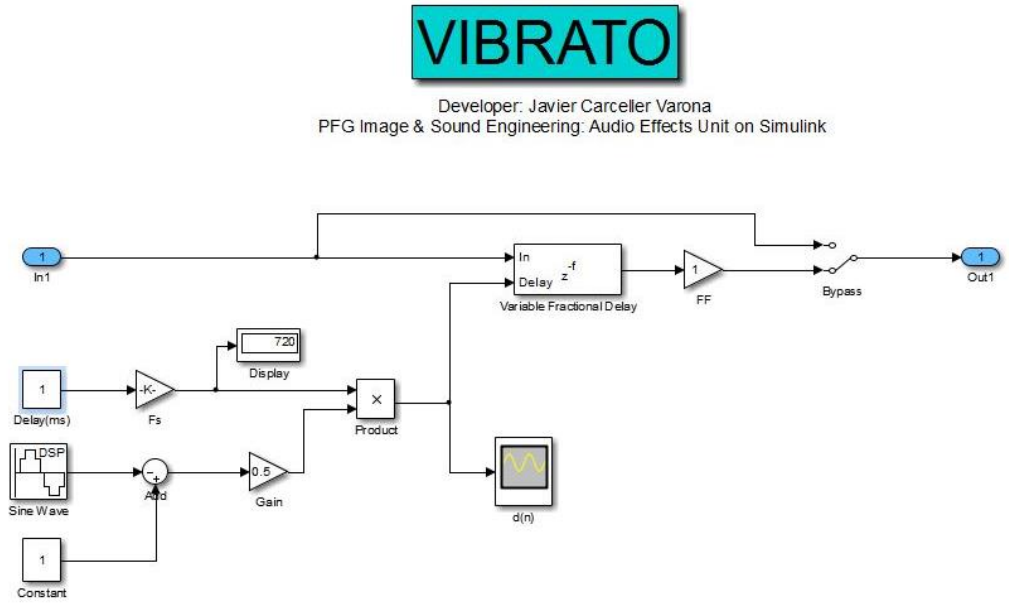

Figura 77: Estructura de un Vibrato en Simulink mediante un filtro paso todo generalizado.

El resultado para una señal sinusoidal de entrada con una frecuencia de 10 Hz se muestra en la figura [78]. Puede visualizarse la fluctuación producida en la frecuencia de la señal por el retardo modulado aplicado en el efecto.

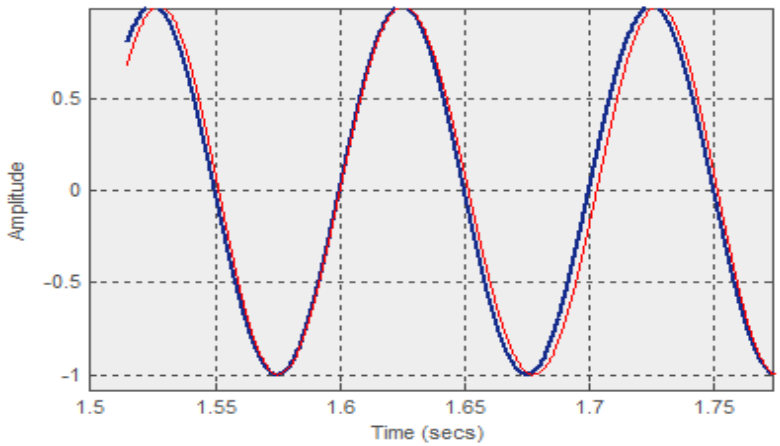

Figura 78: Efecto de Vibrato para una señal sinusoidal de entrada de 10 Hz. Frecuencia de la señal moduladora 5Hz. Retardo D = 144 muestras.

4.4. CONCLUSIONES

Los algoritmos utilizados para los efectos desarrollados son básicos y cumplen con su función. Sin embargo, presentan para determinadas configuraciones de sus parámetros efectos indeseados que colorean la señal aportando sonidos que deberían ser eliminados. En concreto, el mayor problema de utilizar los filtros IIR tipo peine es el realce en altas frecuencias, acompañado de una coloración de la señal que produce un sonido metálico. Este efecto es notable principalmente en el Reverberador, en el *Chorus* y en el *Flanger*, debido a que presentan lazos de realimentación.

En los algoritmos de retardos es importante hablar de la **latencia**, definida como el tiempo de respuesta que tarda el algoritmo en procesar la señal. Esta latencia depende en el fondo del número de operaciones programadas a realizar por el procesador. Uno de los mayores inconvenientes encontrados al desarrollar estos algoritmos en Simulink es la gran latencia existente, ya que se trabajan con retardos muy altos, los cuales requieren un gran número de operaciones. Estas consideraciones deben tenerse en cuenta a la hora de evaluar la calidad del algoritmo. Sin embargo, este problema no se traslada al pasar de código de Matlab/Simulink, mucho más lento, a otros códigos de *plugins* como C, los cuales trabajan a más velocidad y con un menor número de operaciones del procesador, reduciendo en grandes cantidades la latencia.

Los productos comerciales digitales actuales basados en retardos presentan una gran variedad de algoritmos distintos con mejores resultados. El objetivo de un *plugin* de calidad es encontrar el algoritmo ideal que produzca un sonido lo más natural posible, alejado de los matices provocados por el procesado digital. Existen reverberaciones de gran calidad que se asemejan mucho más al sonido real de una sala, como puede ser la reverberación de convolución, o efectos profesionales utilizados en procesadores de audio para instrumentos y voz, con efectos de *chorus, flanger, vibrato*, distintos tipos de ecos, y sinfín de posibilidades en aumento gracias a las nuevas tecnologías y a la capacidad creciente de ordenadores y procesadores de audio.

5. PROCESADO DE DINÁMICA

5.1. INTRODUCCIÓN

Los efectos que engloban el procesado de dinámica son principalmente los **compresores**, **expansores**, **puertas de ruido** y **limitadores**. Se denominan procesadores de dinámica debido a que afectan al rango dinámico de la señal de audio. Los compresores se utilizan para reducir el rango dinámico, mientras que los expansores lo aumentan. Los limitadores y las puertas de ruido son las versiones de los compresores y expansores en el límite, es decir, con el máximo valor de compresión o expansión posible. Sus aplicaciones son muy variadas, desde su uso para radiodifusión adaptando el rango dinámico de la señal al medio por el que se va a transmitir, hasta su uso en grabación y mezcla para atenuar señales de alto o bajo nivel y producir las variaciones que se deseen sobre el rango dinámico.

Existen soluciones analógicas utilizadas durante muchos años, junto con equipos digitales que trabajan únicamente con la dinámica de la señal. Sin embargo, estos equipos han sido reemplazados rápidamente por los procesadores de dinámica en formato *plugin*, mucho más baratos y eficientes. Prácticamente sólo quedan presentes estos efectos en equipos físicos de procesado multiefectos. Los compresores, puertas de ruido y limitadores son utilizados frecuentemente en el mundo del audio en procesos de mezcla y masterización, mientras que los expansores quedan restringidos prácticamente al mundo de la radiodifusión.

Figura 79: Procesador de dinámica 3632 Compressor del fabricante ALESIS.

5.2. FUNCIONES DE TRANSFERENCIA Y ESTRUCTURA DE UN COMPRESOR/EXPANSOR

Los efectos de procesado de dinámica modifican los niveles de la señal de forma continua en función de si se encuentran estos niveles por encima o por debajo del umbral establecido. La expresión básica que aplican estos procesadores se muestra a continuación, siendo x_0 el umbral de compresión/expansión, y ρ la relación de compresión/expansión [ref. 8].

$$y = y_0 \left(\frac{x}{x_0}\right)^{\rho} \quad (72)$$

Aplicando logaritmos, el nivel de salida en función del nivel de entrada se corresponde con la expresión siguiente:

$$20\, log\left(\frac{y}{y_0}\right) = 20\, \rho\, log\left(\frac{x}{x_0}\right) \quad (73)$$

Cuando el procesador es un compresor, si la señal de entrada tiene un nivel superior al umbral x_0, se aplica la relación de compresión ρ con un valor menor a 1 que atenúa la señal, produciendo una disminución del rango dinámico. Si el procesador es un expansor, cuando la señal de entrada es menor a un umbral x_0, se aplica la relación de compresión ρ con un valor mayor a 1 que también atenúa la señal, produciendo un aumento del rango dinámico. En el límite, cuando ρ es mucho menor que 1 (ρ<<1), el compresor actúa como limitador, mientras que cuando ρ es mucho mayor que 1 (ρ>>1), el expansor actúa como puerta de ruido. Representando la función de transferencia descrita anteriormente se puede visualizar de manera sencilla el comportamiento de estos procesadores en función de la señal de entrada, tal y como se muestra en la figura [80]. La relación de compresión/expansión ρ se corresponde con la pendiente de la recta en la zona de aplicación del efecto.

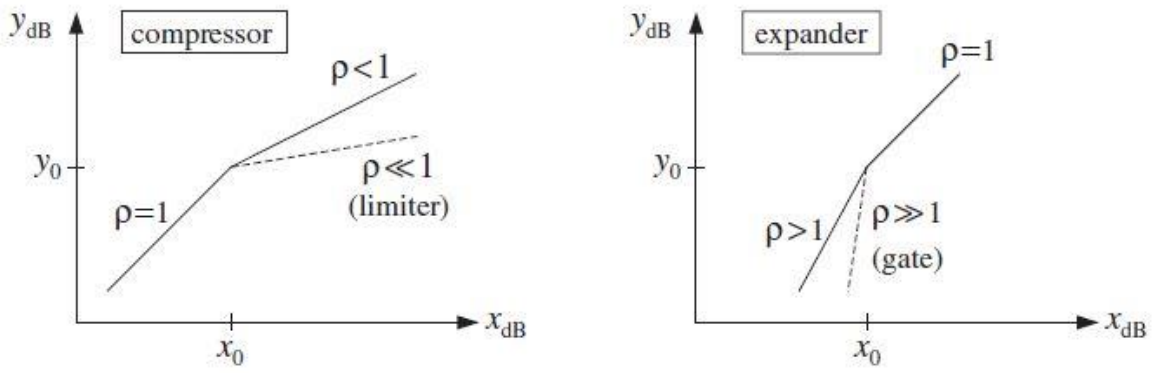

Figura 80: Función de transferencia de un compresor/expansor en función de la señal de entrada.

Existen otra clase de funciones de transferencia, con compresores lineales que presentan una respuesta en forma de recta, o bilineales, con dos zonas diferenciadas de compresión o expansión. Sin embargo, no es común su uso para procesado de audio en estaciones de trabajo.

En tiempo real, la salida del sistema se calcula para cada muestra multiplicando la entrada por una ganancia G, que depende de la relación de compresión/expansión cuando el nivel de la señal de entrada se encuentra en la zona de trabajo del procesador. Desarrollando la expresión [74]:

$$y = y_0 \frac{x^\rho}{{x_0}^\rho} = y_0 \frac{x^{\rho-1}x}{{x_0}^{\rho-1}x_0} = \left(\frac{y_0}{x_0}\left(\frac{x}{x_0}\right)^{\rho-1}\right)x = Gx \quad (74)$$

El valor de la ganancia es, por tanto, siendo G_0 la ganancia umbral:

$$G = G_0\left(\frac{x}{x_0}\right)^{\rho-1} \quad (75)$$

El diagrama de bloques de un compresor/expansor se muestra en la figura [81]. Cada muestra de entrada pasa por un detector de nivel que genera una señal de control denominada c_n, la cual controla la ganancia G_n por la que se multiplica a la muestra de entrada. Dependiendo del tipo de compresor, la señal de control será el valor de pico instantáneo |x(n)|, la envolvente de x(n) o el valor cuadrático medio de x(n). Una forma sencilla de generar la señal de control es utilizar la expresión [76], la cual actúa como detector de envolvente de la señal.

$$c_n = \alpha|x(n)| + (1-\alpha)c_{n-1} \quad (76)$$

Esta ecuación actúa como un rectificador de señal seguido de un filtro paso bajo, donde α es la constante de tiempo que marca el **tiempo de ataque o caída** del compresor/expansor. Se define el tiempo de ataque como el tiempo que tarda el sistema en alcanzar el valor de ganancia teórico cuando el compresor/expansor se activa. El tiempo de caída es el tiempo que tarda en volver a la ganancia unidad cuando el nivel de entrada está fuera del umbral de trabajo. Esta constante tiene el valor de $\alpha = 1/N$, donde N es el número de muestras promediadas por el detector de nivel. La relación entre α y la frecuencia de muestreo se muestra en la ecuación [77], donde $T_{a/c}$ es el tiempo de ataque o caída. Si α = 0, el detector actúa como un detector del nivel de pico de la señal. Es la configuración recomendada para los limitadores de señal, donde se requiere una respuesta del sistema lo más rápida posible para atenuar los niveles superiores al umbral.

$$N = \frac{T_{a/c}}{T_s} = T_{a/c}f_s \rightarrow \alpha = \frac{1}{N} = \frac{1}{T_{a/c}f_s} \quad (77)$$

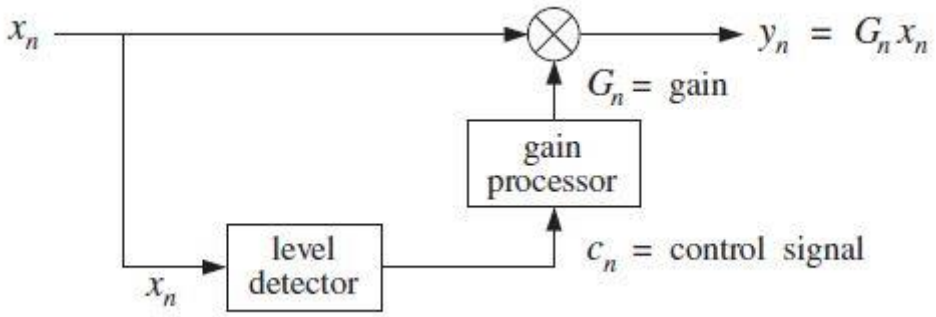

Figura 81: Diagrama de bloques de un Compresor/Expansor.

Finalmente, las expresiones [78] y [79] indican el valor de la ganancia G en función de la señal de entrada. Teniendo en cuenta las consideraciones anteriores, la ganancia es función realmente de la señal de control c_n, por lo que particularizando el cálculo para esta señal en función del umbral de nivel c_0:

- Compresor:

$$\begin{cases} f(c) = \left(\frac{c_n}{c_0}\right)^{\rho-1} & c_n \geq c_0 \\ f(c) = 1 & c_n < c_0 \end{cases} \quad (78)$$

- Expansor:

$$\begin{cases} f(c) = 1 & c_n > c_0 \\ f(c) = \left(\frac{c_n}{c_0}\right)^{\rho-1} & c_n \leq c_0 \end{cases} \quad (79)$$

5.3. DISEÑO DE UN COMPRESOR/LIMITADOR EN SIMULINK

La implementación de un compresor en Simulink se realiza mediante la estructura mostrada en la figura [81]. Cada muestra de la señal de entrada pasa por un detector de envolvente de la señal que genera una señal de control, la cual establece la ganancia a aplicar. Los controles del compresor permiten seleccionar el tiempo de ataque AT, el tiempo de caída RT, el umbral de compresión y el ratio de compresión. Para comprobar el funcionamiento del compresor, se construye una señal sinusoidal pulsante de amplitudes 2 y 0.4, mediante la configuración de bloques mostrada en la figura [82]. Una señal de pulsos de amplitud 0.8 es sumada con una componente continua de 0.2, creando un pulso entre 1 y 0.2. Al multiplicar esta señal por una señal sinusoidal de amplitud 2 se obtiene la señal deseada, mostrada en la figura [83].

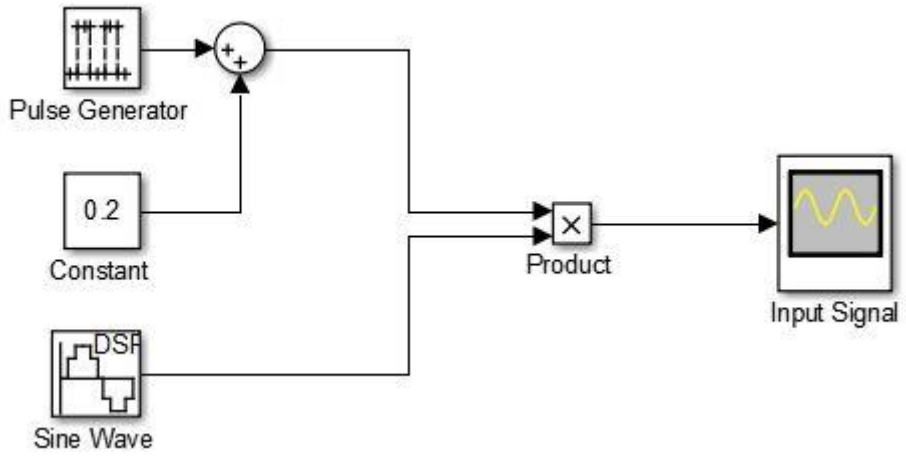

Figura 82: Estructura del generador de la señal de prueba del compresor.

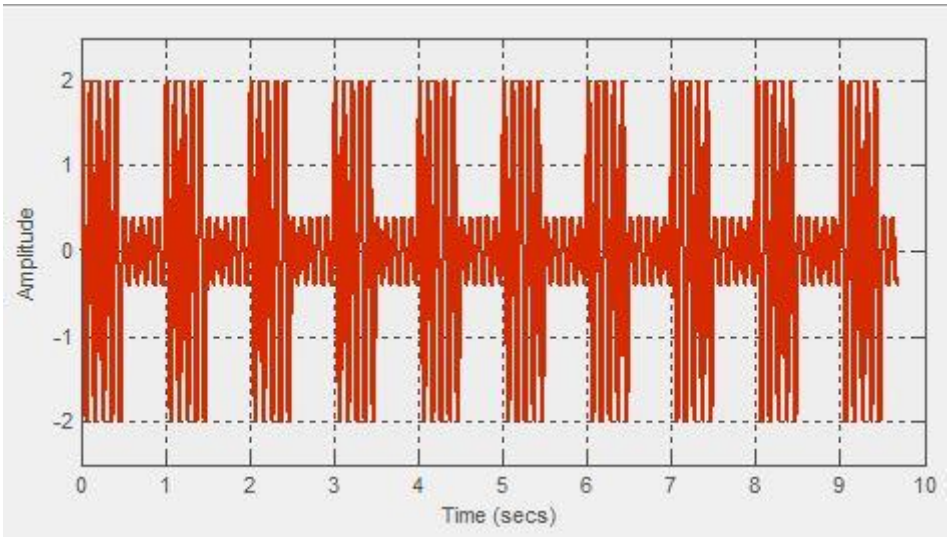

Figura 83: Señal sinusoidal pulsante de prueba para el compresor. Amplitudes 2 y 0.4. Frecuencia: 10 Hz.

La estructura general del compresor se muestra en la figura [84]. Dos subsistemas incluyen las estructuras para calcular la señal de control y la ganancia con los parámetros establecidos a la entrada, calculando α según la expresión [77]. A su vez, el umbral de compresión se calcula en escala lineal mediante la expresión [80]. Visores temporales permiten comprobar la señal de control, la ganancia, y la salida del sistema junto a la entrada.

$$c_0 = 10^{CT_0(dB)/20} \quad (80)$$

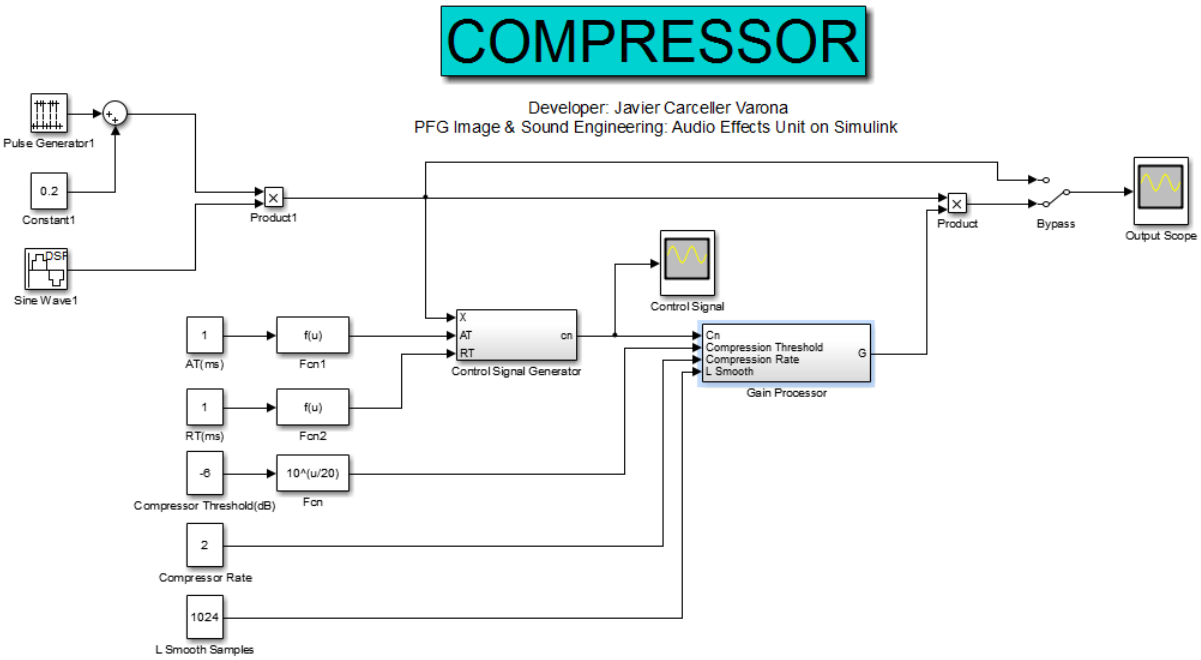

Figura 84: Estructura de un compresor en Simulink.

La señal de control se calcula mediante la expresión [76] descrita en el apartado anterior. Esta expresión puede aplicarse mediante la estructura de bloques siguiente. La estructura está duplicada para el cálculo con el tiempo de ataque AT y el tiempo de caída RT, de forma que la salida del generador de la señal de control es el mayor valor de ambos cálculos. Los bloques de *delay* permiten almacenar en cada iteración el valor de c_{n-1} para el cálculo de la envolvente de la señal.

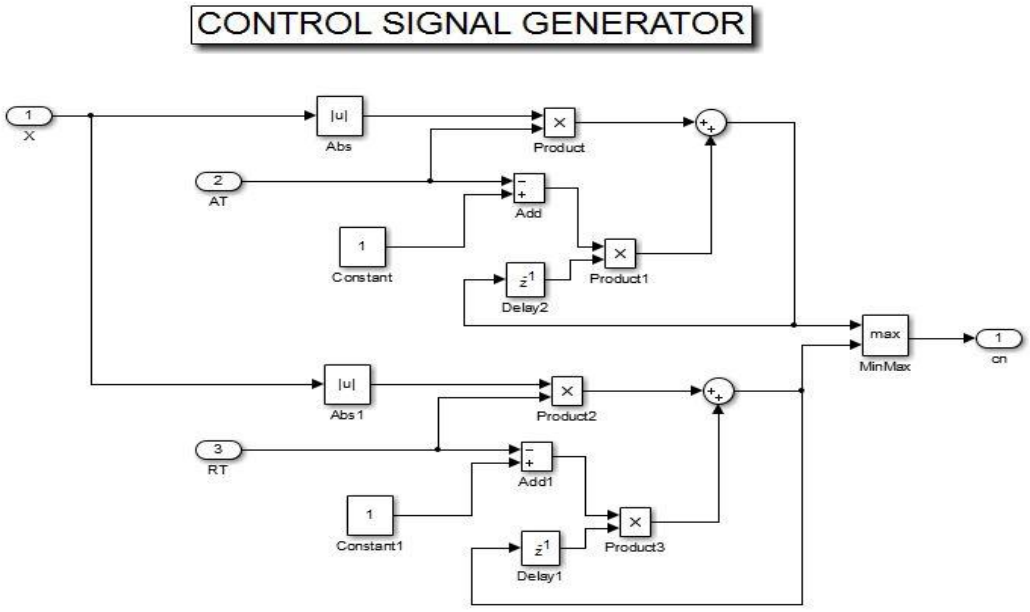

Figura 85: Subsistema generador de la señal de control.

El subsistema generador de la ganancia actúa cumpliendo con la expresión [78] de compresión en función de si la señal de control (el nivel de entrada) se encuentra por encima o por debajo del umbral de compresión c_0. Para realizar esto en Simulink se utiliza un bloque de condición ***if***. Estableciendo como señales de entrada a este bloque la señal de control y el umbral, si la señal de control **u1** es mayor o igual que el umbral de compresión **u2**, se activa el subsistema de condición *if* que genera la ganancia a aplicar a la señal de entrada. En caso contrario, la ganancia de salida es G=1. Esta ganancia se genera mediante un bloque de código de Matlab cuando la señal está por encima del umbral, mostrado a continuación. Debe estar presente el bloque *Action port if* en el subsistema para poder ser activado por el bloque de condición *if* del nivel superior. El bloque final *Merge* entrega a su salida la señal de entrada que ha sufrido modificaciones más recientemente, es decir, el subsistema de condición que ha sido activado previamente.

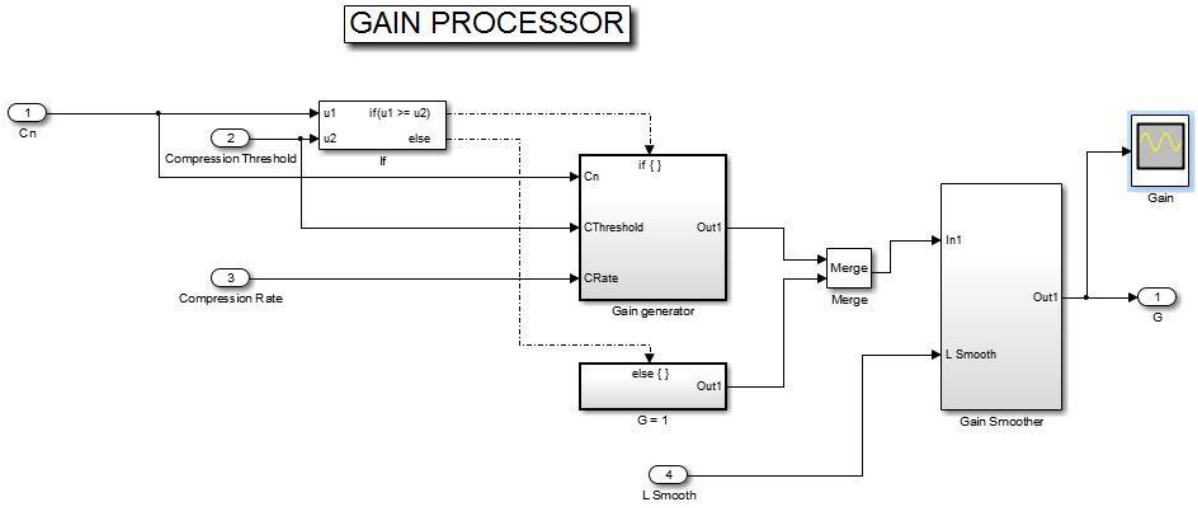

Figura 86: Estructura de procesado de la ganancia en función de la señal de control.

```
function G = GainProcessor(cn,c0,CRate)
%This function calculates the gain to apply to the input signal
%cn: control signal (envelope of the input signal)
%c0: Compressor threshold
%CRate: Compressor rate 1/CRate


G =(cn/c0)^((1/CRate)-1);
```

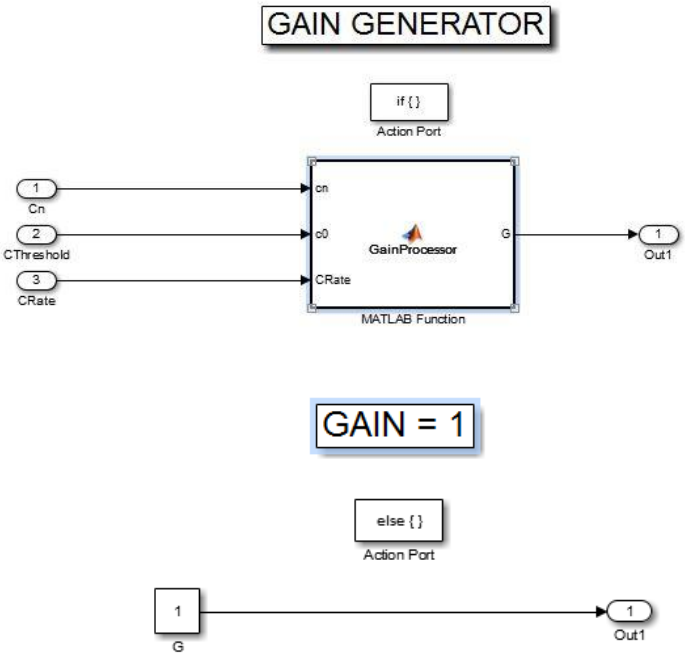

Figura 87: Subsistema de condición if generador de la ganancia del compresor (Arriba). Subsistema de condición if con ganancia unidad (abajo).

La ganancia calculada de esta forma presenta un rizado de magnitud considerable cuando el compresor está en la zona de trabajo. Una forma de mejorar la implementación es reducir el rizado realizando una media móvil de L muestras (en inglés L-point smoother) de la ganancia de la señal, cumpliendo con la expresión [81]. Debe tomarse un valor suficiente de L muestras para conseguir reducir el rizado sin que la media calculada sea demasiado amplia y afecte al comportamiento del compresor. La figura [88] muestra la señal de ganancia con y sin media móvil.

$$G_n = \frac{1}{L} f(c_n) + \left(1 - \frac{1}{L}\right) G_{n-1} \tag{81}$$

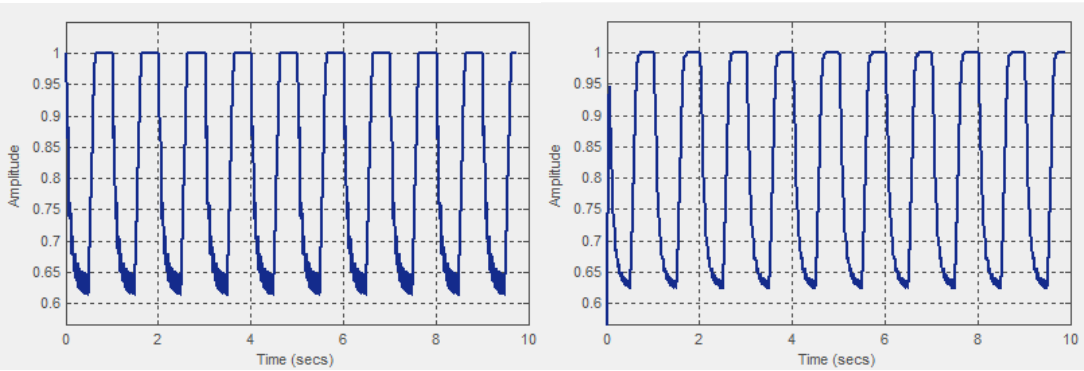

Figura 88: Señal de ganancia sin media móvil (izquierda). Señal de ganancia con una media móvil de L=1024 muestras (derecha).

La expresión anterior se lleva a cabo mediante la estructura mostrada en la figura [89]. Esta estructura tiene una forma muy similar a la utilizada en el cálculo de la señal de control. Se toma como parámetro configurable el número de muestras de la media móvil L.

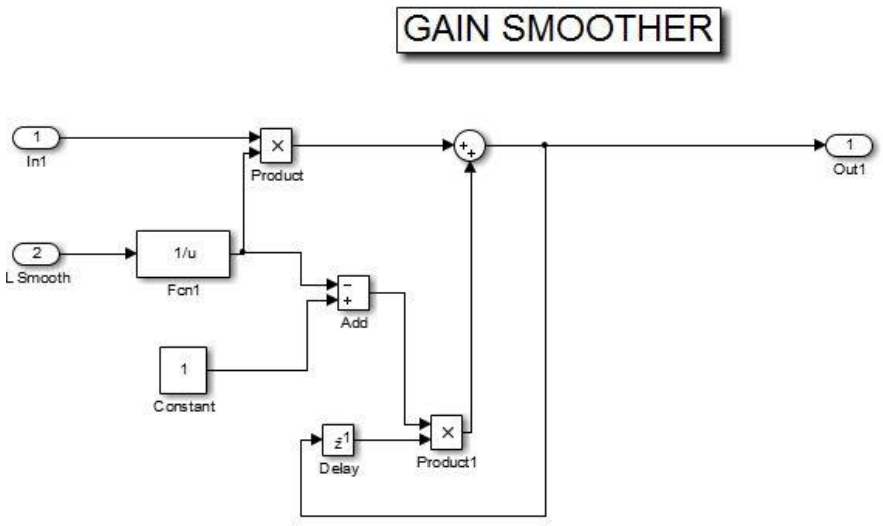

Figura 89: Estructura en Simulink de una media móvil para reducir el rizado de la ganancia.

La figura [90] muestra el resultado de salida del compresor para un umbral de compresión c_0= 0.5 (-6dB) y un ratio de compresión ρ = ½. El tiempo de ataque y caída α se ha definido con un valor de 100ms. La gráfica visualiza cómo la señal sinusoidal superior al umbral de amplitud 2 es atenuada con un ratio de compresión de ½, mientras que la señal sinusoidal por debajo del umbral de amplitud 0.4 queda intacta. Se visualiza también cómo actúa el tiempo de ataque desde que el compresor se activa hasta que alcanza la ganancia adecuada para atenuar la señal sinusoidal por encima del umbral. Se comprueba el efecto del tiempo de caída cuando el compresor se desactiva, hasta que la ganancia alcanza el valor unidad. A su vez, la figura [91] muestra la señal de control y la ganancia aplicada a la señal de entrada, con un valor para la media móvil de L = 1024 muestras.

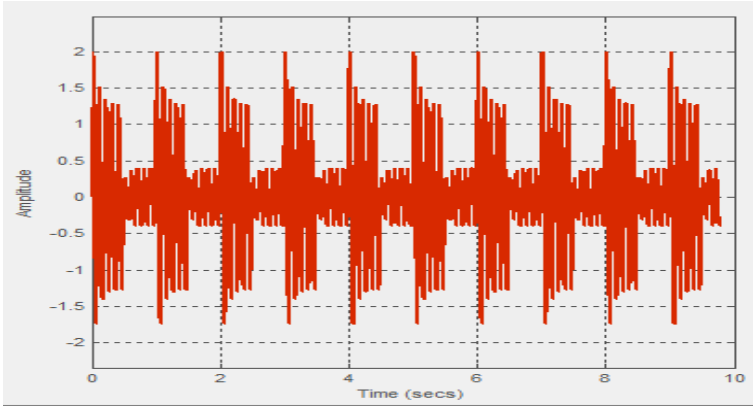

Figura 90: Señal sinusoidal pulsante de salida del compresor. Umbral de compresión C_0=0.5. Tiempo de ataque y caída α=0.1. Ratio de compresión ρ=1/2. Media móvil L =1024 muestras.

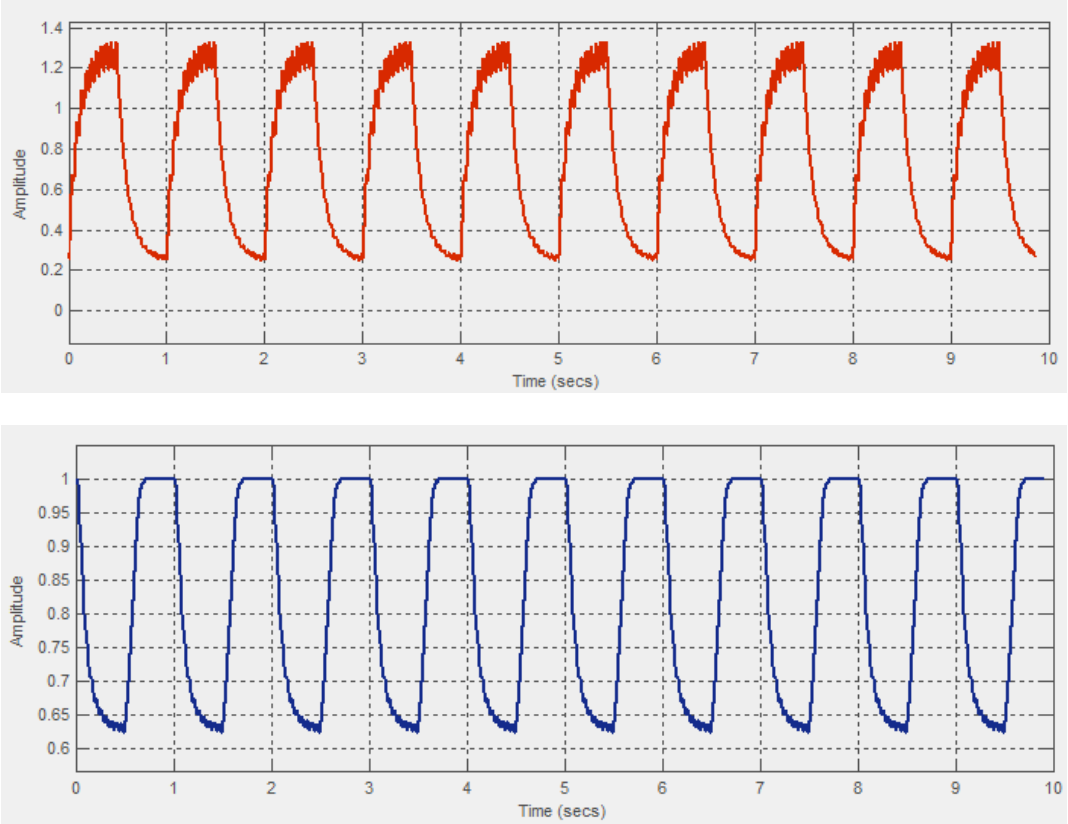

Figura 91: Señal de control: envolvente de la señal de entrada (arriba).Señal de ganancia (abajo). Umbral de compresión C_0=0.5. Ratio de compresión ρ=1/2. Media móvil L =1024 muestras.

Estableciendo un valor del ratio de compresión elevado se consigue que el compresor actúe como un limitador. Para ello es necesario configurar el sistema con un ratio mucho menor que la unidad, normalmente ρ =1/10 o inferior. La figura [92] muestra la señal de salida del compresor con la configuración anterior, estableciendo el ratio de compresión anterior.

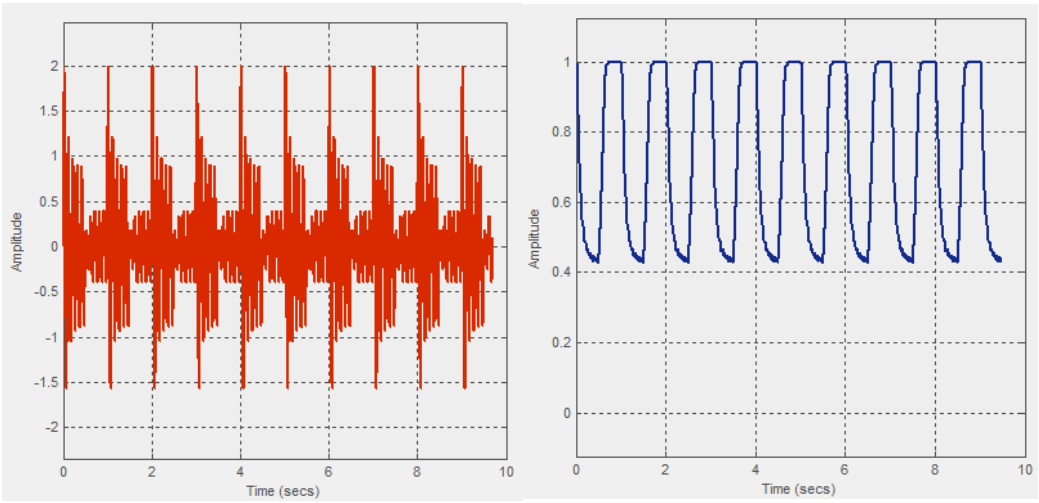

Figura 92: Señal sinusoidal de salida de un limitador (izquierda). Señal de ganancia (derecha). Umbral de compresión C_0=0.5. Ratio de compresión ρ=1/10. Media móvil L =1024 muestras.

5.3. DISEÑO DE UN EXPANSOR/PUERTA DE RUIDO EN SIMULINK

La estructura en Simulink de un expansor tiene exactamente la misma forma que la estructura de un compresor detallada en el apartado anterior. Si bien la diferencia principal es la zona de trabajo del expansor. Cumpliendo con la expresión [79], la relación de expansión se aplica cuando la señal se encuentra por debajo del umbral c_0. La figura [93] muestra la estructura del expansor en Simulink.

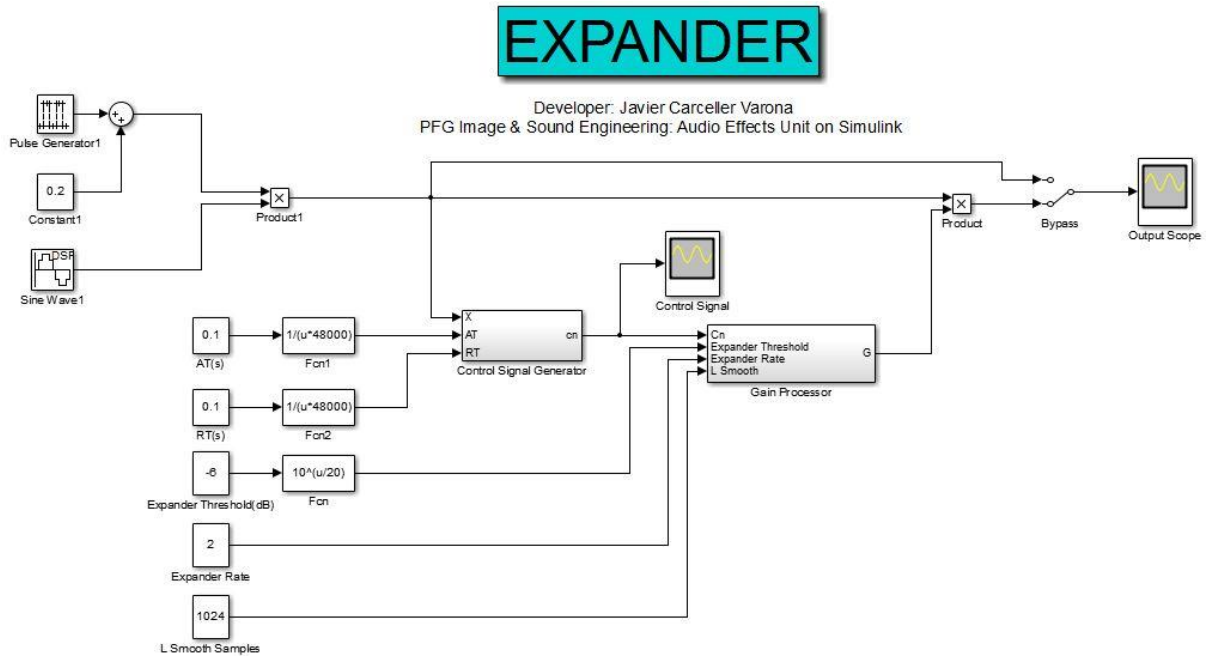

Figura 93: Estructura de un expansor en Simulink.

La figura [94] muestra la principal diferencia con respecto a un compresor. Se ha invertido el bloque de condición if para que la zona de trabajo del expansor se aplique cuando la señal de control está por debajo del umbral. El código de la función en Matlab para generar la señal de ganancia se muestra a continuación.

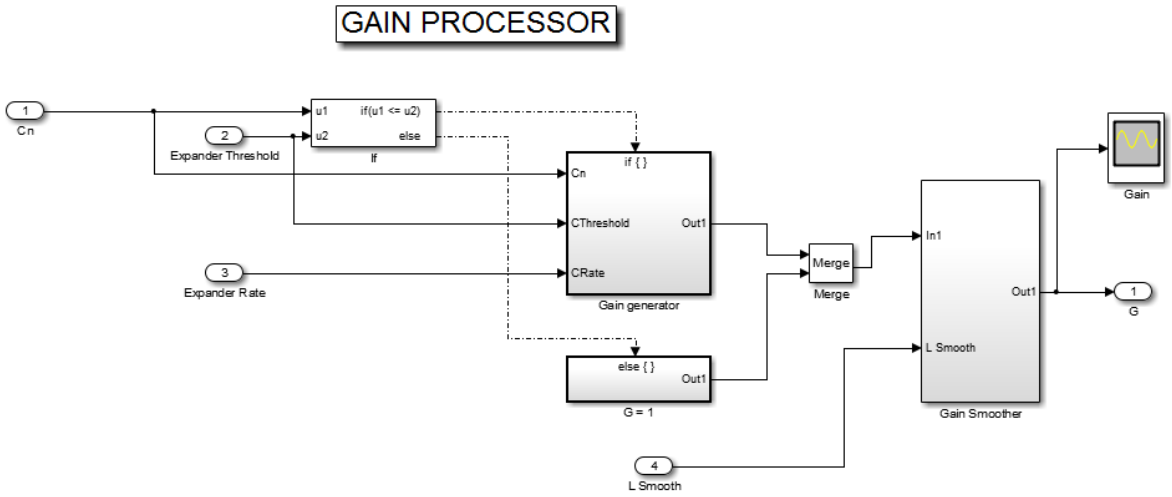

Figura 94: Estructura de procesado de la ganancia en función de la señal de control.

```
function G = GainProcessor(cn,c0,ERate)
%This function calculates the gain to aply to the input signal
%cn: control signal (envelope of the input signal)
%c0: Expander threshold
%ERate: Expander rate

G =(cn/c0)^((ERate)-1);
```

La figura [95] muestra el resultado de salida del expansor para un umbral de expansión c_0= 0.5 (-6dB) y un ratio de expansión ρ = 2. El tiempo de ataque y caída α se ha definido con un valor de 100ms. La gráfica visualiza cómo la señal sinusoidal inferior al umbral de amplitud 0.4 es atenuada con un ratio de expansión de 2, mientras que la señal sinusoidal por encima del umbral de amplitud 2 queda intacta.

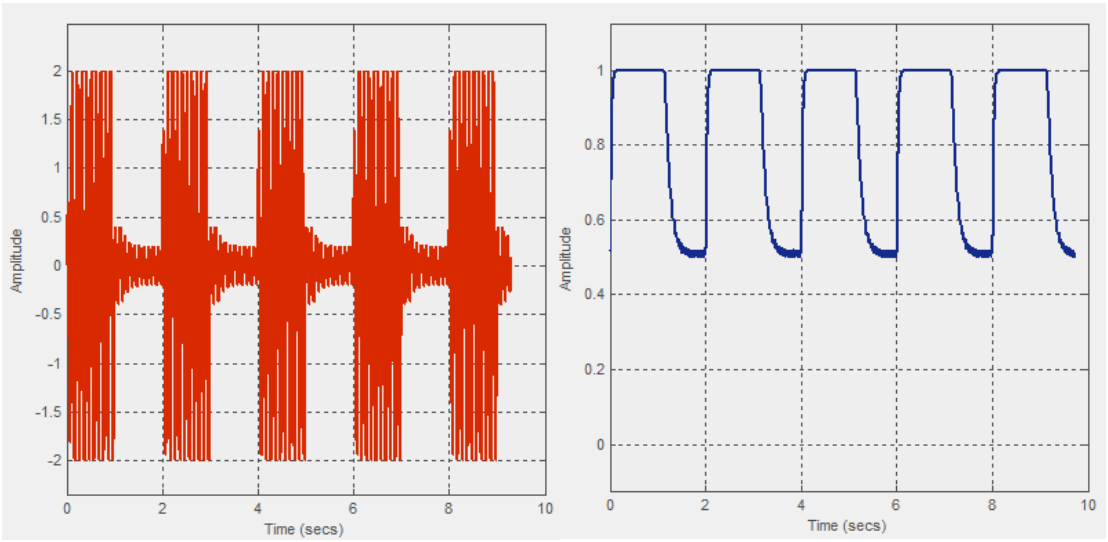

Figura 95: Señal sinusoidal pulsante de salida del expansor (izquierda).Señal de ganancia (derecha). Umbral de expansión C0=0.5. Tiempo de ataque y caída α=0.1. Ratio de expansión ρ=2. Media móvil L =1024 muestras.

Estableciendo un valor del ratio de expansión elevado se consigue que el compresor actúe como una puerta de ruido. Para ello es necesario configurar el sistema con un ratio mucho mayor que la unidad, normalmente ρ =10 o superior. Las puertas de ruido son muy útiles en el mundo del audio, ya que como su propio nombre indican permiten eliminar secciones de la pista de audio con niveles por debajo de un determinado umbral. Un ejemplo de aplicación es la presencia de ruido de un instrumento en las secciones de silencio de una pieza musical. La figura [96] muestra la señal de salida del expansor con la configuración anterior, estableciendo el ratio de expansión anterior.

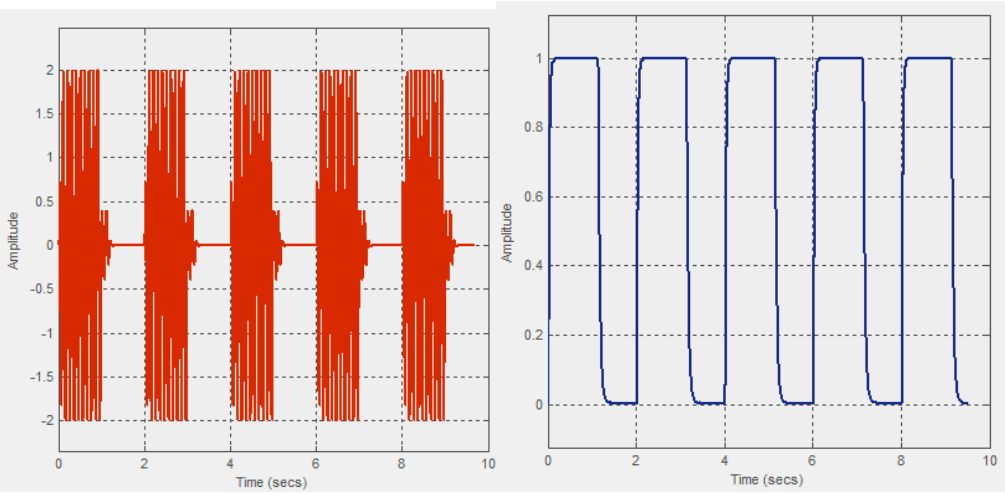

Figura 96: Señal sinusoidal de salida de una puerta de ruido (izquierda). Señal de ganancia (derecha). Umbral de compresión C_0=0.5. Ratio de compresión ρ=1/10. Media móvil L =1024 muestras.

5.4. CONCLUSIONES

El diseño de estos algoritmos en Simulink es un proceso relativamente sencillo con una gran ventaja respecto a otro tipo de software. La potente herramienta de visualización de señales que presenta Simulink con sus múltiples configuraciones permite realizar un seguimiento detallado de las señales utilizadas en el algoritmo de compresión y expansión, verificando el funcionamiento correcto del algoritmo de forma muy rápida y sencilla.

En este algoritmo se ha demostrado también la capacidad de los bloques de activación, como son los bloques de condición *if*. Aunque la implementación utilizada es muy sencilla, Simulink presenta una gran cantidad de bloques que pueden ser activados o condicionados de múltiples formas, incluyendo a su vez varios niveles de programación. Esta posibilidad da paso a poder realizar programaciones de alto nivel al igual que en código Matlab u otros lenguajes de programación, con la gran ventaja de que puede

establecerse de forma sencilla e intuitiva los activadores de estos bloques con señales utilizadas en la simulación.

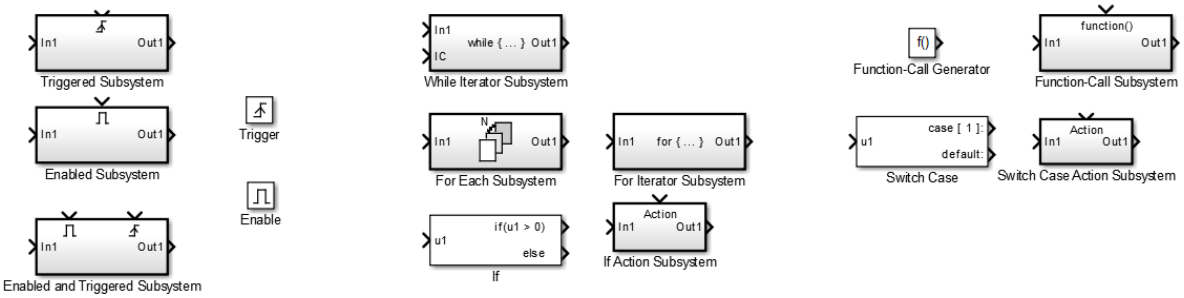

Figura 97: Bloques de condición y activación de Simulink. Library: Simulink/Ports & Subsystems.

En cuanto al algoritmo en sí, la principal desventaja son los sobreimpulsos que aparecen cuando se activa el compresor, los cuales colorean la señal de audio y pueden originar distorsiones. Además, debe realizarse correctamente el control de la media móvil para reducir el rizado de la ganancia. Un número reducido de muestras provoca que la señal de ganancia presente un rizado elevado cuando el efecto está en la zona de trabajo. Un número elevado de muestras reduce al mínimo el rizado de la señal de ganancia. Sin embargo, el número de muestras con el que se realiza la media afecta a un intervalo demasiado elevado, por lo que interviene sobre el funcionamiento del efecto, tal y como se muestra en la figura [98]. Esto puede provocar un sonido indeseado en la señal de audio a tratar.

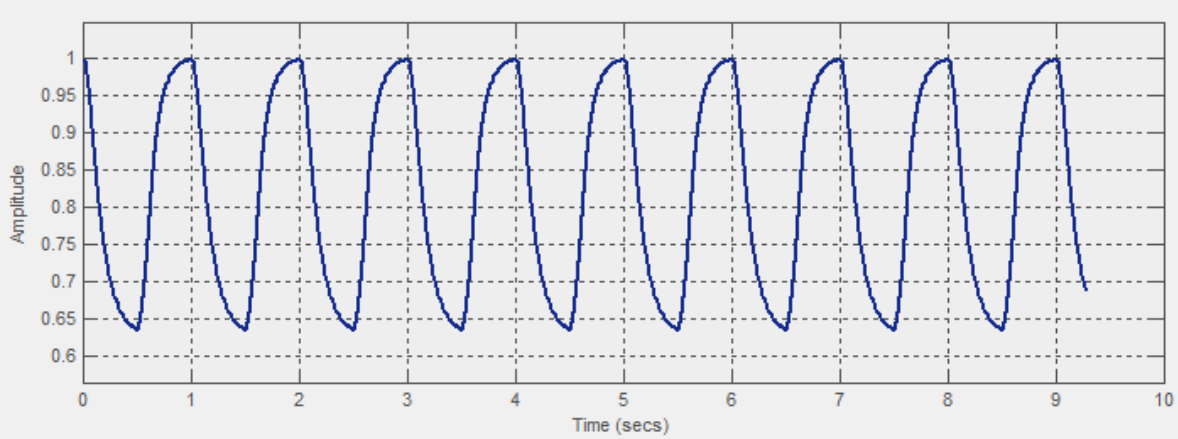

Figura 98: Señal de ganancia para una media móvil de L = 4096 muestras.

6. COMPRESIÓN-EXPANSIÓN TEMPORAL

6.1. INTRODUCCIÓN

Los algoritmos de compresión y expansión temporal permiten realizar un cambio de escala de la señal de audio. Estos cambios son posibles tanto en el dominio del tiempo como en el dominio de la frecuencia, produciendo dos tipos de efectos muy utilizados actualmente, denominados ***Pitch Shifting*** y ***Time Stretching***. Los algoritmos de modificación de la escala temporal o *time stretching* pretenden comprimir o expandir la duración de una señal de audio sin producir modificaciones en frecuencia. Esto es, evitando cambios de tono de la señal a tratar. Por otro lado, este último concepto es el buscado por los algoritmos de modificación en frecuencia o *pitch shifting*. Su objetivo es mover en frecuencia la señal de audio para así producir modificaciones en el *pitch* de la señal de audio.

Se define el *pitch* de una señal de audio como el rango de frecuencias en el que se encuentra. En el caso de señales de voz o *speech signals*, el *pitch* se corresponde con la frecuencia fundamental de la voz generada. La señal de voz presenta una periodicidad medible junto con una serie de armónicos producidos por el tracto vocal y que caracterizan la voz de cada individuo.

La utilización de estos algoritmos abarca un amplio abanico de temáticas diferentes, muchas de las cuales se encuentran fuera de los procesos de audio musical, donde se utilizan para corregir errores en el tempo de una grabación, errores de tono de un instrumento o voz, o para añadir efectos artísticos a una pieza musical o mezcla de canciones por compositores o *Djs*. Fuera del mundo de audio musical, algoritmos en este aspecto son utilizados por *vocoders* y aplicaciones de captación de voz, libros de audio grabado, etc…

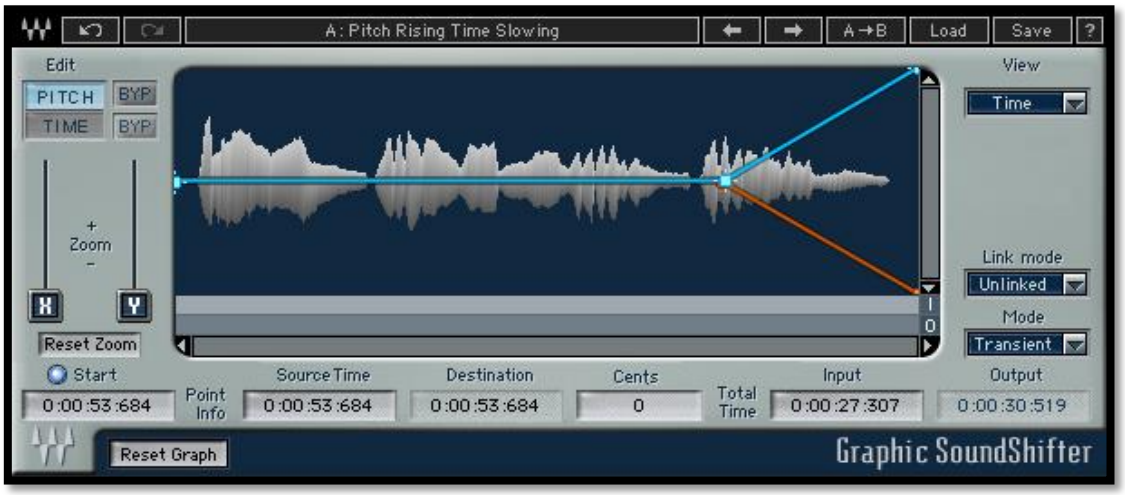

Figura 99: Time Stretching & Pitch Shifting plugin VST de la marca WAVES.

6.2. VELOCIDAD DE REPRODUCCIÓN VARIABLE

Atendiendo a los conceptos de procesado digital, la manera más intuitiva y rápida de conseguir un efecto de compresión y expansión temporal o modificación del *pitch* es realizar un cambio de la frecuencia de muestreo de la señal de audio a tratar. Este proceso se traduce en una **velocidad de reproducción variable** de la señal de audio. En un primer acercamiento, si la frecuencia de muestreo f_{s2} aumenta con respecto a la frecuencia de muestreo inicial f_{s1}, la señal de audio se acelera, por lo que se comprime en el tiempo y aumenta el tono, produciendo un sonido más agudo. Si por el contrario, la frecuencia de muestreo f_{s2} disminuye con respecto a la frecuencia de muestreo inicial f_{s1}, la señal de audio se ralentiza, por lo que se expande en el tiempo y disminuye el tono, produciendo un sonido más grave.

En un sistema de procesado digital donde la señal entrante y saliente es analógica, la frecuencia de muestreo a la que trabajan el conversor A/D y el conversor D/A debe ser la misma, por lo que la forma de realizar el cambio de muestreo es la utilización de un algoritmo de procesado digital intermedio, tal y como se muestra en la figura [100].

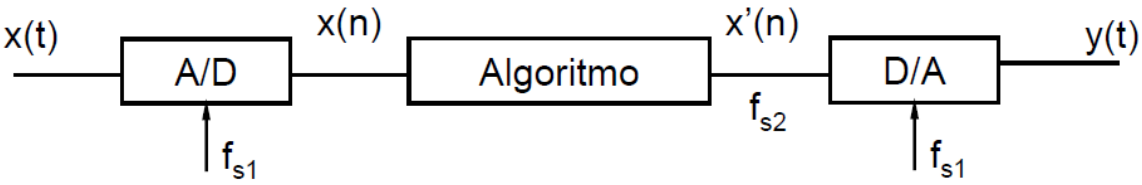

Figura 100: Sistema digital con algoritmo modificador de la frecuencia de muestreo.

En procesado digital, existen dos métodos principales para producir modificaciones con factores no enteros en la frecuencia de muestreo, proceso denominado ***resampling***. Estos son la **interpolación** y el **diezmado**, descritos en el apartado de introducción [1.5].

6.3. TIME STRETCHING

Como se ha explicado previamente, el algoritmo de *time stretching* permite ajustar una pista de audio en el tiempo que se desee sin modificar el pitch de la señal, realizando un cambio de la frecuencia de muestreo. Existen dos procesos fundamentales para conseguir este efecto. Son los algoritmos **OLA** (*overlap-add*), y su versión mejorada **SOLA** (*synchronous overlap-add*).

6.3.1. OVERLAP-ADD

El algoritmo *Overlap-add* permite realizar un cambio en la frecuencia de muestreo en un factor no entero mediante la concatenación y suma de fragmentos de la señal de audio. Para ello, la señal se divide en fragmentos de N muestras, con una determinada región de solape entre cada fragmento de S_i muestras. Posteriormente, los fragmentos son sumados realizando un desplazamiento de los mismos en un factor $S_0 = x S_i$, donde x es el factor de compresión o expansión de la señal de audio deseado. La figura [101] muestra un esquema de la separación en segmentos de la señal y la suma de estos.

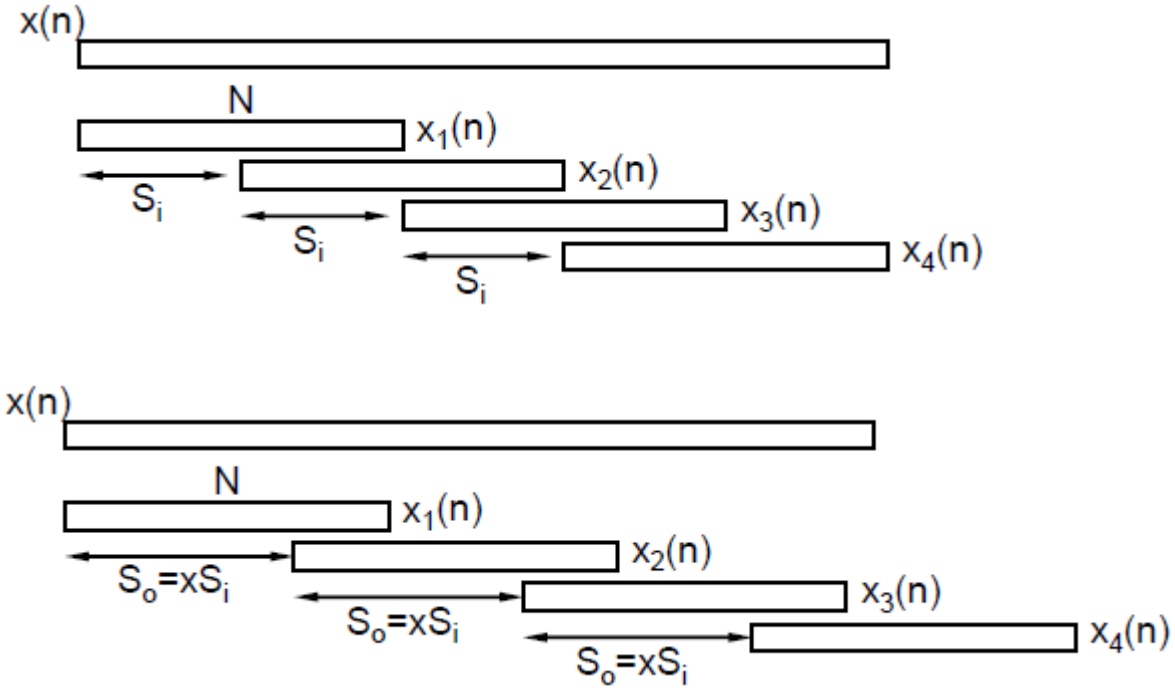

Figura 101: Algoritmo Overlap-add. Separación en segmentos de N muestras cada S_i muestras (arriba). Concatenación de los segmentos desplazados en un factor x (abajo).

Si el factor x es mayor que 1 (x > 1), se reduce el solape entre segmentos con respecto al solape inicial, por lo que al ser concatenados se expande la duración de la señal de audio. Si por el contrario el factor x es menor que 1 (x < 1), aumenta el solape entre segmentos con respecto al solape inicial, por lo que la señal de audio se comprime, disminuyendo su duración.

A la hora de concatenar los segmentos, se realiza un *cross-fade* de ambos segmentos en la región donde se solapan las muestras, de forma que se realice una suma ponderada gracias a los procesos de *fade-in* y *fade-out*, tal y como se muestra en la figura [102].

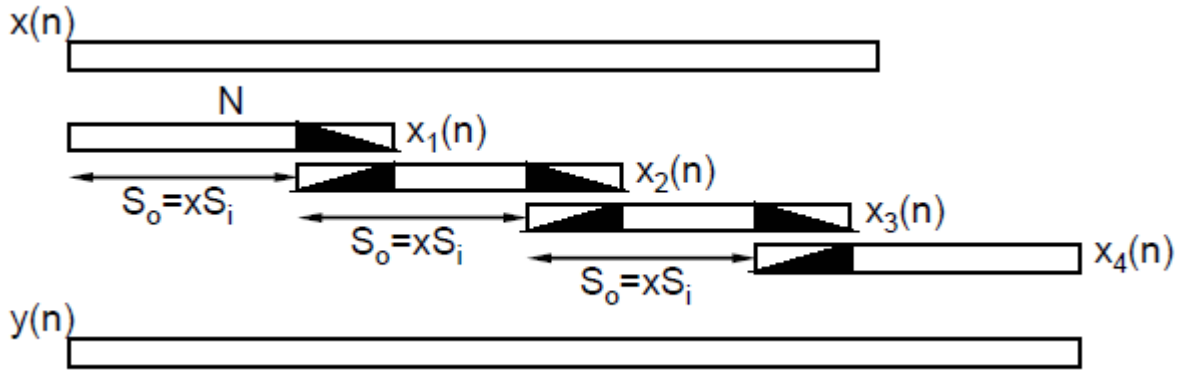

Figura 102: Cross-fade en la región de solape para concatenar los segmentos.

Este algoritmo es un sistema muy básico que presenta un gran inconveniente. Cuando las regiones donde se solapan los segmentos son muy dispares muestra a muestra, pueden producirse cambios de fase que generan *clips* en la señal de audio. Los *clips* deterioran en gran medida la señal y deben evitarse. Además, produce un efecto de reverberación que colorea la señal, lo que hace este algoritmo inviable en ocasiones en las que se requiere buena calidad de audio.

6.3.2. SYNCHRONOUS OVERLAP-ADD

Concebido como una mejora del algoritmo anterior, soluciona los problemas de cambio de fase al concatenar los segmentos deslizando el punto de concatenación [ref. 1]. Los pasos a seguir por el algoritmo son los siguientes:

- Divide la señal en segmentos de N muestras cada S_i muestras.
- Desplaza los segmentos en un factor $S_0 = x\ S_i$.
- Busca el punto de máxima similitud entre ambas regiones de solape de cada pareja de segmentos a concatenar mediante el cálculo de la correlación cruzada, que sigue la expresión [82]. L es el número de muestras de solape, x_1 y x_2 son los segmentos a concatenar.

$$r_{x1,x2} = \frac{1}{L}\sum_{n=0}^{L-m-1} x_1(n)x_2(n+m), \qquad 0 \le m \le L \tag{82}$$

- Se obtiene el valor en el tiempo discreto donde la correlación es máxima, denominado k_m.
- Manteniendo el punto de máxima correlación en el medio del *cross-fade*, se aplica el *fade-in* y el *fade-out* a ambos segmentos a concatenar, respectivamente. La figura [103] muestra un esquema de este proceso.
- Finalmente, se aplica la concatenación (*overlap-add*) de los segmentos, obteniendo la señal de audio final expandida o comprimida.

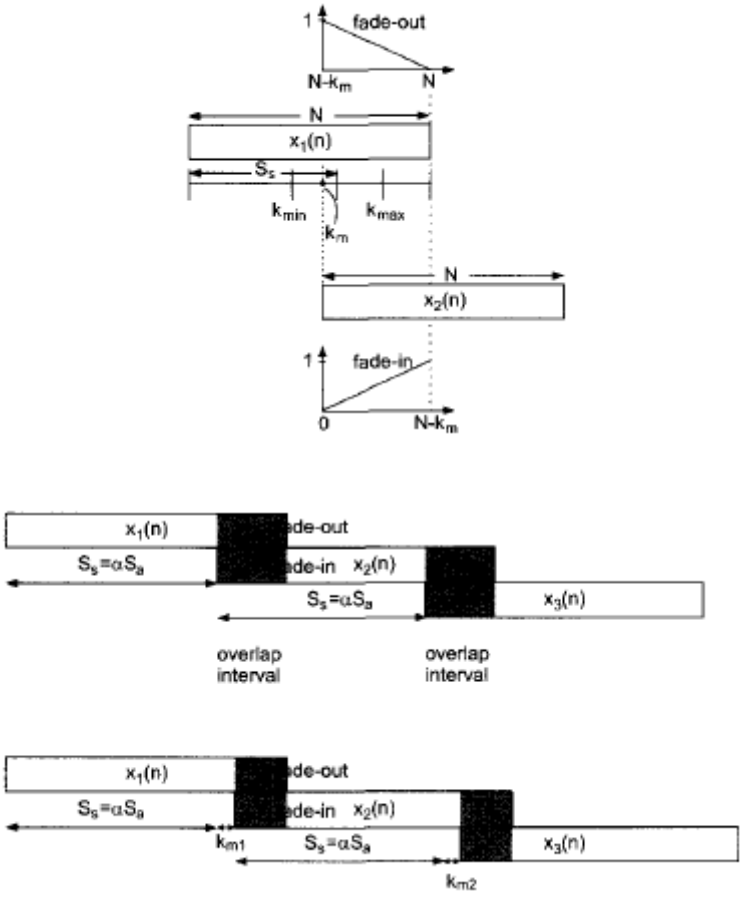

Figura 103: Proceso del Synchronous Overlap-Add.

6.3.3. DISEÑO EN MATLAB DEL ALGORITMO TIME STRETCHING

La principal dificultad del algoritmo *SOLA* es que no se trata de un proceso en tiempo real. Esto quiere decir que para realizar su implementación en Simulink, sería necesario conocer de antemano los tamaños de las variables y señales modificadas, por lo que es muy complejo realizar una simulación con un flujo de señal continuo en Simulink. Aun sabiendo los valores de S_i y S_0, habría que procesar todo el archivo para encontrar los puntos de correlación máxima, algo inviable para archivos de audio ya que se trata de una gran cantidad de muestras y operaciones a realizar.

Por tanto, la forma más rápida e intuitiva de implementar el algoritmo de *SOLA* es utilizar el propio código de Matlab. Los parámetros de configuración principales son los controles del número de muestras de cada segmento y el rango de solape, S_i y N. La función acepta como parámetro de entrada el factor de compresión o expansión temporal x (alpha en el código para evitar confusión con los términos). Se establece un intervalo de búsqueda del punto de máxima correlación fijo, con un valor de L = 256*alpha/2.

El código sigue los pasos anteriores. Separa la señal en segmentos, y procesa las parejas de segmentos calculando el punto de máxima correlación, donde aplica el *cross-fade* mediante un *fade-in* y un *fade-out*. Seguidamente realiza la suma de ambas zonas solapadas y procesadas por el *cross-fade*, para construir el segmento que será utilizado en la siguiente iteración. El código en Matlab se muestra a continuación.

```
function [x1]= SOLA(x,alpha)
%This function escalates the time of the input signal with a synchronous
%overlap-add algorithm.
%x: input signal
%alpha: time scaling factor
%Developer: Javier Carceller Varona
%PFG Image & Sound Engineering: Audio Effects Unit on Simulink


Si =256;% Analysis hop size
N = 4096; %block length
M = ceil (length(x)/Si);%Calculates the number of segments
L=round(256*alpha/2);%Overlap interval for searching the max correlation

S0= round(Si*alpha);%Time shift
xmax=zeros(1,M-1);%Vector with the max correlations of the segments
index=zeros(1,M-1);%Index of the max correlations
x(M*Si+N)=0;
x1= x(1:N);%First Segment

for ni= 1:M-1
    x2 = x(ni*Si+1:N+ni*Si);%Second Segment
    Xcorrelation = xcorr(x2(1:L),x1(1,S0*ni: S0*ni +(L-1)));%Calculates
de              cross- correlation of the two segments
    [xmax(1,ni),index(1,ni)] = max(Xcorrelation); %Max correlation
    fadeout=1:(-1/(length(x1)-(ni*S0-(L-1)+index(1,ni)-1))):0;%Cross-
fade
    fadein=0:(1/(length(x1)-(ni*S0-(L-1)+index(1,ni)-1))):1;%Cross-fade

    Tail= x1(1,(ni*S0-(L-1))+index(1,ni)-1:length(x1)).*fadeout;%Fade-
out interval of the first segment
    Begin=x2(1:length(fadein)).*fadein;%Fade-in interval of the second
segment
    Add= Tail+Begin;%Overlap-add
    x1= [x1(1,(1:ni*S0-L+index(1,ni)-1)) Add
x2(length(fadein)+1:N)];%Next segment to repeat the process
end
```

La figura [104] muestra el resultado al aplicar el algoritmo sobre una señal de voz, con un factor de expansión de 1.2. A su vez, se muestra en la figura [105] el resultado al aplicar un factor de compresión de 0.8. Debido al proceso de *overlap-add*, los niveles de la señal quedan afectados, siendo diferentes de los niveles de la señal original.

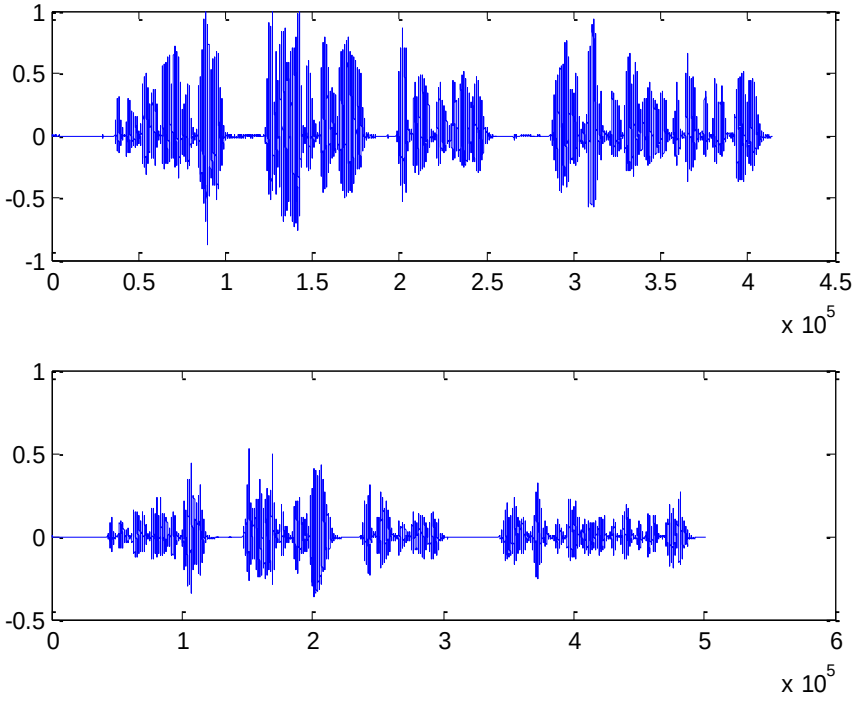

Figura 104: Señal de voz original (arriba). Señal expandida en el tiempo por un factor de expansión de 1.2 (abajo).

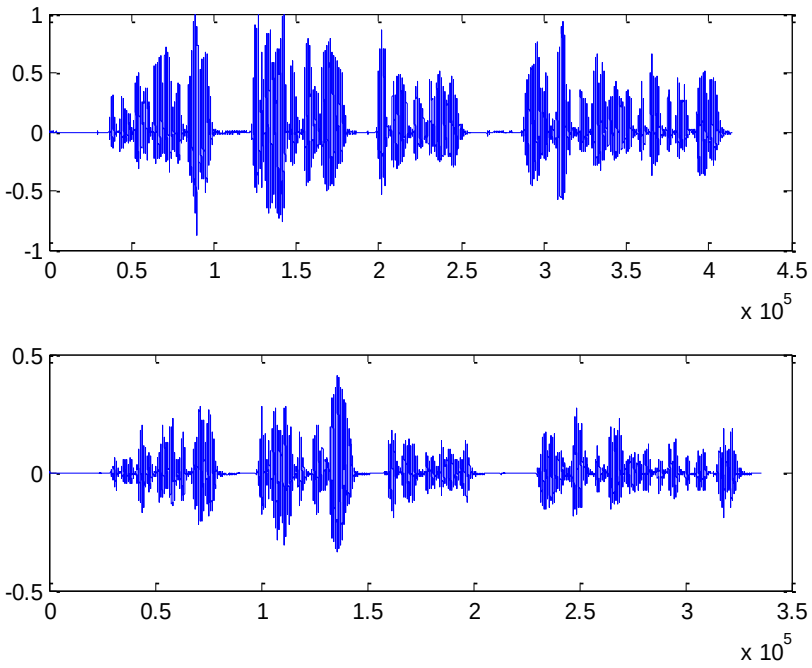

Figura 105: Figura 74: Señal de voz original (arriba). Señal comprimida en el tiempo por un factor de compresión de 0.8 (abajo).

6.4. PITCH SHIFTING

Los algoritmos de cambio de *pitch*, como se ha descrito anteriormente, se utilizan para modificar el tono o *pitch* de una señal de audio sin alterar su duración en el tiempo. La forma más sencilla de realizar este proceso es utilizar el algoritmo ***SOLA*** de *time stretching* descrito previamente junto con un proceso de remuestreo de la señal o *resampling*. El remuestreo de la señal permite modificar el *pitch* en función del factor deseado. Este proceso produce a su vez una compresión o expansión temporal, al introducir o eliminar muestras mediante los procesos de interpolación y diezmado. Para corregir este efecto se utiliza el algoritmo *SOLA*, realizando la compresión o expansión necesaria de la señal en función del factor de remuestreo, para así devolver la señal de audio a su duración original.

Si el proceso de remuestreo se realiza con un factor N_1/N_2, la señal se comprime o expande según este factor. Para volver al número de muestras inicial, el algoritmo *SOLA* debe realizarse con un factor inverso de N_2/N_1 [ref. 1].

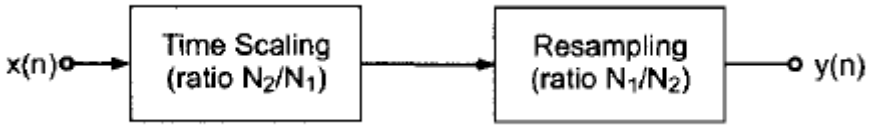

Figura 106: Proceso de pitch shifting mediante el algoritmo SOLA.

6.4.1. DISEÑO EN MATLAB DEL ALGORITMO PITCH SHIFTING

De nuevo, la forma más sencilla de implementar este algoritmo es el uso de código Matlab, ya que este proceso no conlleva un flujo continuo de señal. Para el proceso de *time stretching* se utiliza la función de Matlab *SOLA.m* descrita en el apartado [6.3.3].

El proceso de resampling puede realizarse de dos maneras distintas. Matlab incluye la función denominada de la misma forma que permite remuestrear la señal en un factor N_1/N_2. Para realizar el proceso de forma didáctica y permitiendo visualizar cada uno de los pasos, se ha optado por codificar una función propia que permita este proceso, siguiendo los pasos explicados en el apartado de introducción [1.5].

Para procesar ficheros de larga duración, la señal se divide en ventanas o *frames*, siguiendo un funcionamiento similar a los *frames* de Simulink. Cada uno de estos *frames* pasa por el proceso de interpolación, filtrado paso bajo para evitar los solapes espectrales y realizar la interpolación de la señal y diezmado. Las funciones utilizadas para los

procesos de interpolación y diezmado son *interp(x, N_1)* y *decimate(x, N_2)*, donde N_1 y N_2 son los factores de interpolación y diezmado respectivamente.

El proceso de filtrado se realiza mediante la implementación de un filtro FIR paso bajo, con orden seleccionable mediante la variable N y frecuencia de corte según la expresión [83]. El filtro se construye mediante la función de Matlab *fir1(N-1, ω_c),* que entrega los coeficientes de un filtro FIR de orden N (N+1 coeficientes). La frecuencia de corte debe estar comprendida entre 0 y 1, siendo el valor para 1 la mitad de la frecuencia de muestreo. Por ello, el valor de la frecuencia de corte se toma sin tener en cuenta la constante π. La ganancia del filtro se corresponde con el valor de interpolación L.

$$\omega_c = min\left[\frac{\pi}{L}, \frac{\pi}{M}\right] = min\left[\frac{1}{N_1}, \frac{1}{N_2}\right] \quad (83)$$

La gráfica [107] muestra la respuesta en frecuencia del filtro FIR configurado para un orden de N=32, y una frecuencia de corte de π/2 rad. A continuación se muestra el código completo de la función en Matlab.

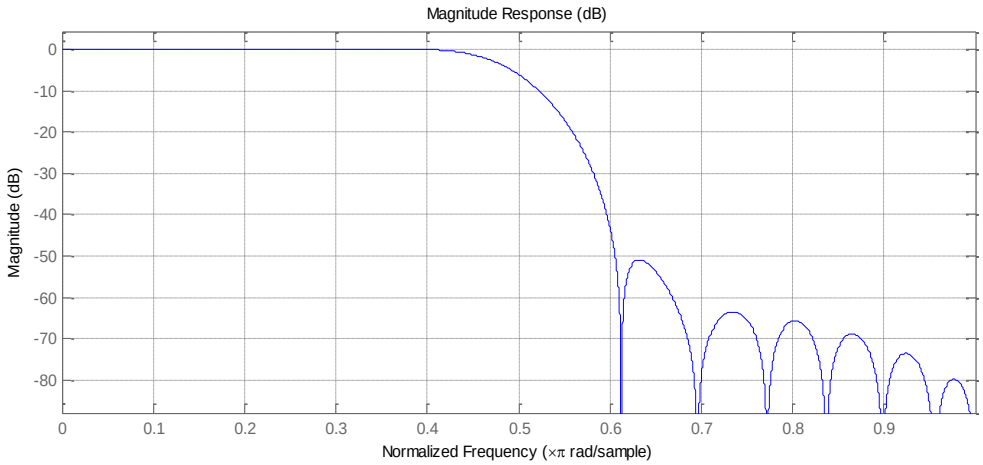

Figura 107: Filtro FIR paso bajo de orden N=32 y frecuencia de corte ω_c=π/2.

```
function y= pitchshifting_sola(x,n1,n2)
%This function changes the pitch of the input signal in a ratio equal to
%n1/n2
%x: input signal
%n1/n2: pitch scaling factor
%Developer: Javier Carceller Varona
%PFG Image & Sound Engineering: Audio Effects Unit on Simulink

    N=32;%Fir filter size
    size=2048;%Frame size for resampling
    R= ceil(length(x)/size);%Number of frames
    wc=min(1/n1,1/n2);%Cutt-off frequency of the FIR filter
    b=fir1(N-1,wc);%Coeff for a low-pass FIR filter
```

```
    %MAIN LOOP
    for ni=1:R-1
        frame= x(ni*size+1-size:size*ni);
        x_interp= interp(frame,n1);%Interpolation
        x_filtered=filter(b,1,x_interp);%Filtering
        x_decim=decimate(x_filtered,n2);%Decimation
        y(ni*length(x_decim)+1-
length(x_decim):ni*length(x_decim))=x_decim;
    end
    y = SOLA(y,n2/n1);%Time-Stretching process
end
```

6.5. CONCLUSIÓN

Los algoritmos descritos de *Time Stretching* y *Pitch Shifting* son algoritmos rudimentarios que consiguen estos efectos, pero con resultados no profesionales. La señal de salida de audio al realizar cualquiera de estos efectos presenta una serie de *clips* audibles y otros artificios, que los convierten en poco recomendables para su uso con la señal de audio, sobre todo si la señal es musical. Son por tanto efectos destinados a señales de voz utilizando pequeñas modificaciones con factores en un rango entre el 1% y el 20% con respecto a la señal original.

Como se ha comentado previamente, existe una mayor complejidad añadida a la hora de realizar el proceso de compresión-expansión temporal en Simulink mediante el algoritmo *SOLA*, debido a que Simulink requiere especificar las dimensiones de las señales dentro del programa. El proceso de búsqueda del punto de mayor correlación para cada fragmento del método *Overlap-Add* requiere de bucles de programación atendiendo a la muestras de solape de forma concreta, proceso complicado en Simulink ya que se trabaja con *frames*. Por tanto, la forma más sencilla de implementar estos procesos es mediante el código en Matlab. Esto conlleva un gran inconveniente. Son algoritmos con un tiempo lento de ejecución para archivos de audio de duraciones por encima de 10 segundos, debido al gran número de muestras a frecuencias de muestreo de 44,1kHz en adelante.

Estos algoritmos, sin embargo, suponen la base para una serie de efectos más complejos y desarrollados, que permitan obtener resultados profesionales de alta calidad. Comprender esta base da lugar a entender el proceso digital necesario para efectos de compresión y expansión temporal u otros efectos derivados.

7. VOCODER DE FASE

7.1. INTRODUCCIÓN

A pesar de que el objetivo del *Vocoder* de fase es, en este proyecto, la compresión y expansión temporal descrita en el capítulo 6, se ha incluido un capítulo aparte para describir el funcionamiento y aplicación que conlleva este algoritmo de procesado digital de la señal. Esto se debe a que la estructura principal de un *Vocoder* de fase es común, independientemente de la aplicación, realizando un procesado intermedio en función del efecto que se desee conseguir.

Los *Vocoders* pueden definirse como un sistema de procesado digital basado en la técnica de **análisis – síntesis** [ref. 4]. El sistema entrega a su salida una señal idéntica a la señal de entrada, o que ha pasado por un procesado capaz de cambiar alguna de sus características. Por tanto, este proceso es muy útil a la hora de realizar cambios de pitch y compresiones o expansiones temporales, consiguiendo un resultado prácticamente libre de artificios. En el caso de *Vocoders* de fase, el proceso de análisis consiste en modelar la señal de entrada como una suma de sinusoides sin relación entre sí, cada una con su amplitud, su frecuencia y su fase. Esto permite trabajar con cualquier señal, tanto voz como señales musicales, al no existir relación de tipo armónica entre las sinusoides.

Existen fundamentalmente tres interpretaciones posibles de los *Vocoder* de fase. La primera es la descrita anteriormente como **suma de sinusoides**. Otra forma de interpretar los *Vocoder* de fase es como un **banco de filtros** paso banda, cada uno de los cuales presenta una amplitud y frecuencia central distinta. La forma de implementar este banco de filtros no es igual que la de un banco de filtros común. Es necesario utilizar bancos de filtros **heterodinos**. En este proyecto, por su mejor rendimiento se ha optado por la tercera interpretación, consistente en la **transformada corta de Fourier** (*Short-Time Fourier Transform* o *STFT*), que será explicada con detalle posteriormente.

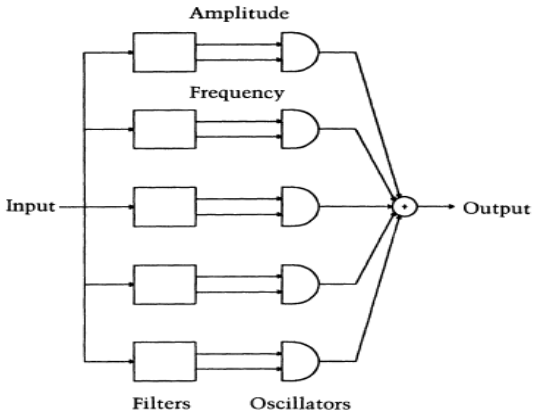

Figura 108: Interpretación mediante un banco de filtros heterodinos.

Una de las aplicaciones actuales del *Vocoder* de fase más extendida es el *plugin* profesional *Auto-Tune*, propiedad de *ANTARES AUDIO TECHNOLOGY*, capaz de modificar el pitch de una señal hasta llegar al semitono deseado, de forma que puedan corregirse defectos a la hora de realizar grabaciones de voz o actuaciones en vivo, ya que puede funcionar en tiempo real.

Figura 109: Plugin Auto-Tune propiedad de ANTARES AUDIO TECHNOLOGY.

7.2. IMPLEMENTACIÓN MEDIANTE TRANSFORMADAS DE FOURIER

Tal y como se ha explicado previamente, los *Vocoder* de fase interpretan la señal de entrada como una suma de sinusoides. Por tanto, existe una relación directa entre esta interpretación y la teoría de Fourier, la cual expone que toda señal periódica puede ser descompuesta en una suma de señales sinusoidales. Es importante relacionar este proceso con el banco de filtros paso banda. Mientras que la interpretación del banco de filtros se centra en visualizar cada una de las frecuencias en el tiempo, la implementación mediante la transformada de Fourier permite analizar todo el margen de frecuencias con su amplitud y su fase en cada instante de tiempo. Esta idea se muestra de forma gráfica en la figura [110] [ref. 4].

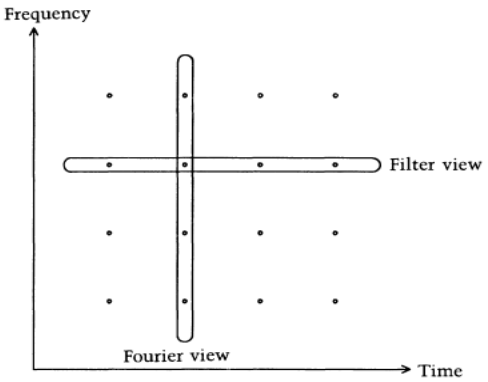

Figura 110: Representación de las interpretaciones de un Vocoder de fase mediante un banco de filtros heterodinos y mediante transformadas de Fourier.

Por supuesto, aunque la señal de entrada presente cierta periodicidad, no es común utilizar señales perfectamente periódicas. En cualquier caso real, la señal presenta un periodo arbitrario que puede variar a lo algo de su propia duración, alejando al sistema de la suposición de que puede ser descompuesta en una suma de señales sinusoidales. Por ello cobra especial importancia la **fase** de la señal. Gracias a la transformada corta de Fourier puede realizarse esta separación en sinusoides con señales no periódicas. El proceso de enventanado permite realizar la suposición de que el fragmento de la ventana se repite indefinidamente, aplicando a posteriori la transformada de Fourier de cada fragmento.

Para poder comprender cómo funciona realmente un *Vocoder* de fase, es necesario observar su estructura, mostrada en la figura [111] [ref. 1]. La señal de entrada que pasa por el proceso de análisis es en primer lugar dividida en fragmentos enventanados con ventanas solapadas entre sí. Posteriormente se aplica la transformada de Fourier mediante transformadas rápidas o **FFT**, obteniendo una serie de *frames* del espectro de la señal enventanada, tanto en módulo como en fase. Llegados a este punto, la señal se encuentra en el dominio del **tiempo-frecuencia**. Es en este punto donde se realiza el procesado determinado del módulo y/o la fase de cada *frame* en función del tipo de efecto a conseguir.

A continuación, es necesario realizar una correcta reconstrucción de la señal, para así obtener de nuevo una señal en el tiempo. Para ello, a partir del módulo y la fase procesados se calcula la transformada inversa de Fourier mediante la **IFFT**, culminando a continuación el proceso mediante la aplicación del método ***Overlap-add***. Esto es debido a que el proceso de enventanado con solape modifica el periodo de muestreo de la señal, pudiendo corregirse este defecto mediante una interpolación. Como ya se ha descrito en el apartado anterior [6.3.1], el método *Overlap-Add* permite realizar un cambio del periodo de muestreo de la señal, consiguiendo que la señal de salida se encuentre a la misma frecuencia de muestreo que la señal de entrada.

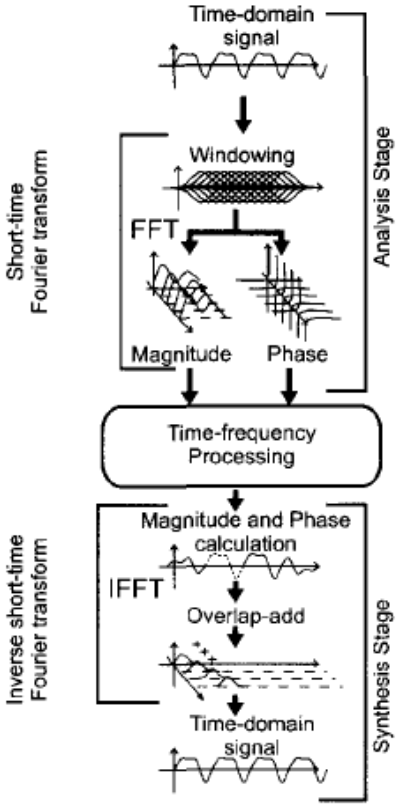

Figura 111: Proceso de un vocoder de fase mediante transformadas cortas de Fourier.

Es importante para evitar *Aliasing* en el proceso de síntesis utilizar también ventanas tras realizar la IFFT. La elección de la ventana a utilizar en ambos procesos es importante, ya que el uso de ventanas rectangulares genera ruido y distorsiones no deseadas. Además, el proceso de *Overlap-Add* depende en gran medida de la ventana escogida, introduciendo un cambio en la ganancia final de la señal. En este caso, por sus propiedades se ha optado por utilizar la ventana de ***Hanning*** [ref. 4]. A la hora de escoger el tamaño de la ventana, tamaños elevados permiten una mayor resolución espectral. No obstante, debe tenerse en cuenta que para realizar correctamente el efecto deseado y no perder información de la señal, es necesario realizar una gran cantidad de *STFTs* a lo largo del tiempo.

La gráfica [112] [ref. 4] muestra el comportamiento de las ventanas de *Hanning* tras el proceso de *Overlap-add* dependiendo de la relación entre el tamaño de la ventana escogida y el intervalo de solape o *hop size*. Esta gráfica muestra que valores del intervalo de solape mayores a 1/2 de la longitud de la ventana producen modulación de la señal. El intervalo que da mejores resultados se obtiene para un valor del 50% o inferior, como por ejemplo entre 1/4 y 1/3 de la longitud de la ventana.

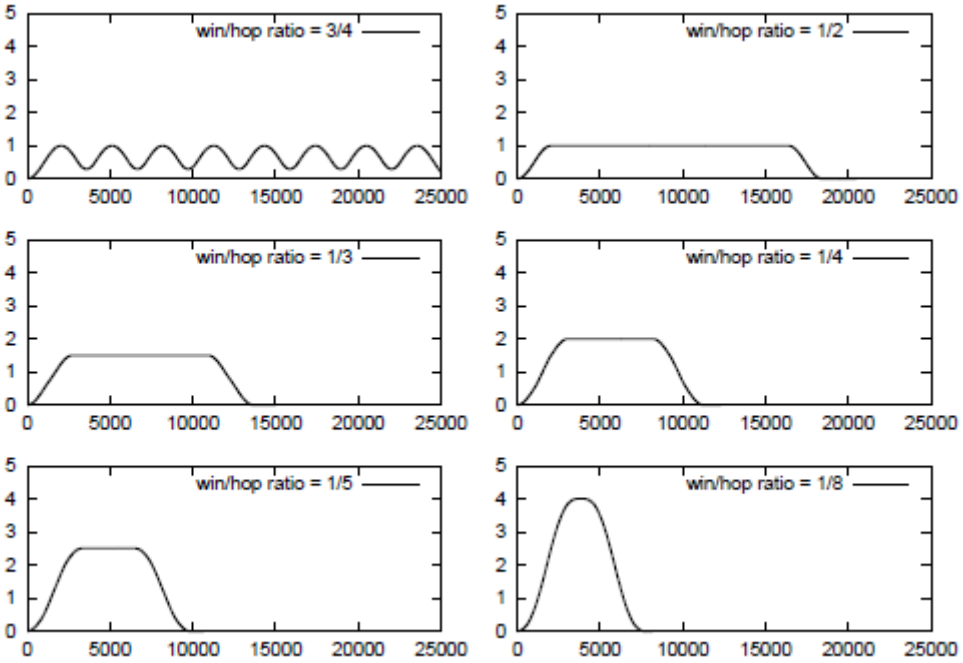

Figura 112: Ventanas de Hanning tras el proceso de Overlap-add.

Como se ha explicado previamente, al tratar con señales que no son perfectamente periódicas, cobra especial importancia la fase. El proceso de cálculo de la fase provoca saltos de 2π radianes cuando ésta excede el rango [-π, π], entregando valores de fase que no se corresponden con el valor real. Estos saltos deben eliminarse para el correcto procesado de la fase y reconstrucción de la señal mediante el proceso de *unwrapped phase*. La reconstrucción de la fase para cada punto de la *STFT* se realiza mediante la expresión [84], donde k indica la frecuencia en la *STFT* y n el tiempo. Para cada frecuencia, la fase se obtiene mediante la fase „*sin envolver*" o *unwrapped phase* sumada a la fase de la frecuencia anterior. La *unwrapped phase* Δφ se define como la diferencia de fase existente entre un punto de la *STFT* y el punto siguiente y la fase correspondiente a la frecuencia nominal del fragmento, y se calcula mediante la expresión [85]. Se obtiene sumando la desviación de fase $\tilde{\varphi}_d$ al incremento nominal de fase entre cada punto, Ω_k, donde N es el cociente entre el intervalo de solape del proceso de análisis y la longitud de la ventana [ref. 1].

$$\tilde{\varphi}(n,k) = \tilde{\varphi}_{n-1} + \Delta\varphi \tag{84}$$

$$\Delta\varphi(n,k) = \underbrace{\frac{2\pi k}{N}}_{\Omega_k} n + \tilde{\varphi}_d \qquad con\ N = \frac{hop}{wLen} \tag{85}$$

Los valores de todas las variables de fase deben estar dentro del intervalo [-π,π], por lo que es necesario utilizar una función denominada ***principal argument***, de forma que los valores arbitrarios de fase se encuentren dentro de este rango. Esta función

cumple con la expresión [86]. La gráfica [113] muestra la salida de esta función dependiendo de la fase de entrada.

$$\varphi = mod(\varphi_{in} + \pi, -2\pi) + \pi \qquad con - \pi < \varphi < \pi \tag{86}$$

De esta forma:

$$\varphi_x = princarg[2\pi m + \varphi_x] \qquad con\ m = 0,1,2 \ldots y\ -\pi < \varphi_x < \pi \tag{87}$$

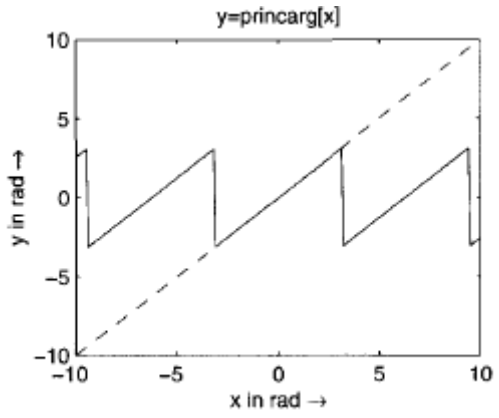

Figura 113: Función Principle Argument dependiendo de la fase de entrada x.

Este proceso de cálculo de fase requiere una explicación matemática más detallada. Si denominamos $\tilde{\varphi}$(s) a la fase de un punto de la *STFT* y $\tilde{\varphi}$(s+1) a la fase del punto siguiente, la fase a calcular para la reconstrucción se obtiene a partir de la fase del punto anterior según la expresión [88], donde Ω_k es la frecuencia nominal del fragmento.

$$\tilde{\varphi}_t(s+1) = \tilde{\varphi}(s) + \Omega_k \tag{88}$$

Para calcular la *unwrapped phase* [85], se suma a la fase de la expresión anterior la desviación de fase, calculada mediante la expresión [89].

$$\tilde{\varphi}_d(s+1) = princarg[\tilde{\varphi}(s+1) - \tilde{\varphi}_t(s+1)] \tag{89}$$

$$\tilde{\varphi}_u(s+1) = \tilde{\varphi}_t(s+1) + \tilde{\varphi}_d(s+1) \tag{90}$$

Derivando la expresión anterior se obtiene la diferencia de fase $\Delta\varphi$:

$$\Delta\varphi(s+1) = \tilde{\varphi}_u(s+1) - \tilde{\varphi}(s) = \Omega_k + princarg[\tilde{\varphi}(s+1) - \tilde{\varphi}(s) - \Omega_k] \tag{91}$$

Una vez obtenida la diferencia de fase *unwrapped*, el cálculo para la reconstrucción de la fase del fragmento actual de la *STFT* se realiza mediante la expresión [85] descrita anteriormente.

Finalmente, el proceso de *Overlap-add* al sumar los intervalos de solape produce una variación significativa en la ganancia de la señal de salida. La expresión [92] aplica una atenuación a la señal de salida para contrarrestar este efecto [ref. 4], en función del intervalo de solape.

$$y = \frac{x}{\frac{\sum_{0}^{wLen-1} hanning(wLen)^2}{synthesis\_hop}} \quad (92)$$

7.3. IMPLEMENTACIÓN DE LA INTERPRETACIÓN MEDIANTE STFT EN SIMULINK

Con el proceso descrito anteriormente, es sencillo realizar la estructura en Simulink gracias al flujo de señal mediante enventanado. La gráfica [114] muestra la estructura del nivel superior. Se presentan tres señales de entrada distintas de prueba para comprobar la correcta reconstrucción de la señal, aunque puede ser configurado para tomar cualquier otro tipo de señal de entrada. La primera es una señal sinusoidal de frecuencia 10Hz y amplitud 1. El archivo „*speech.wav*" es un archivo de voz, mientras que el archivo „*test.wav*" es un archivo musical. El selector presente a continuación selecciona un único canal de la señal de entrada para los casos en los que la señal sea estéreo. A la salida se utiliza el bloque *signal to workspace* para exportar la señal de salida a Matlab y así poder guardarla en un archivo „*wav*". Se establece también un bloque de salida al dispositivo de reproducción para poder escuchar el resultado. Los parámetros de configuración se establecen en el bloque *Phase Vocoder* mediante una máscara. Estos parámetros son la longitud de la ventana y los intervalos de solape tanto en el proceso de análisis como en el proceso de síntesis.

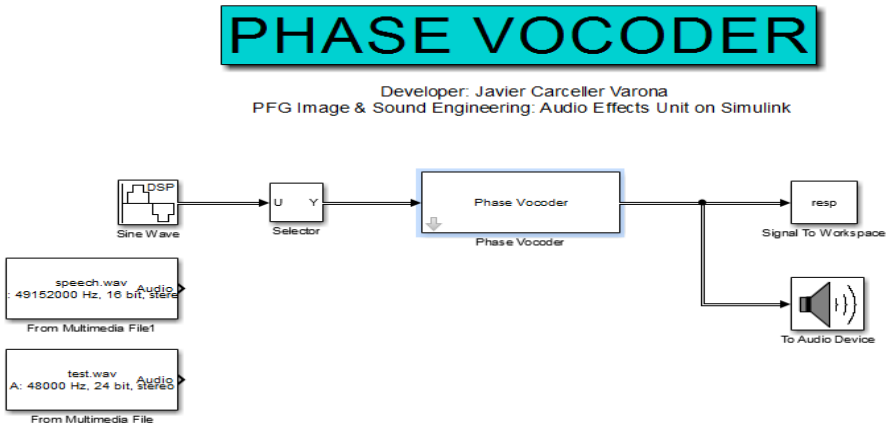

Figura 114: Estructura superior en Simulink del Vocoder de fase.

Es importante configurar el bloque *To Audio Device* para reproducir la señal de salida con la misma frecuencia de muestreo que la señal de entrada, ya que seleccionando la opción por defecto Simulink puede interpretar que la salida tiene una frecuencia de muestreo diferente. La estructura interna del *Phase Vocoder* se muestra en la figura [116]. Este subsistema presenta tres subsistemas distintos con los procesos de análisis, procesado de la señal y síntesis descritos anteriormente.

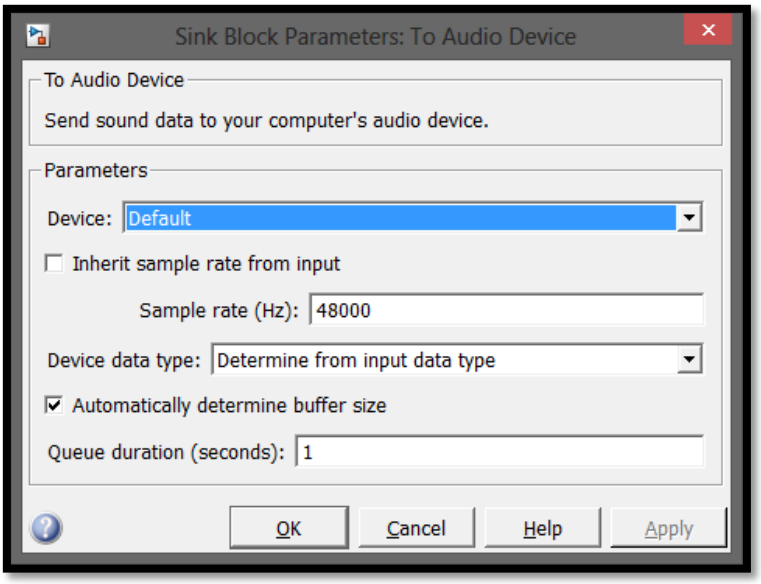

Figura 115: Ventana de configuración del bloque To Audio Device.

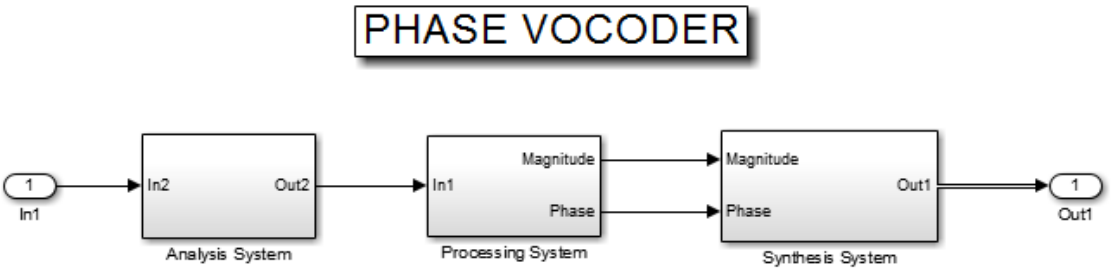

Figura 116: Subsistema Phase Vocoder con los bloques de análisis, procesado de la señal y síntesis.

El bloque de análisis se realiza mediante un buffer que divida la señal de entrada en muestras de longitud *wLen*, tomando intervalos de solape de *wLen-ana_hop*. El fragmento es enventanado mediante una ventana *Hanning*, para a posteriori realizar la FFT del fragmento.

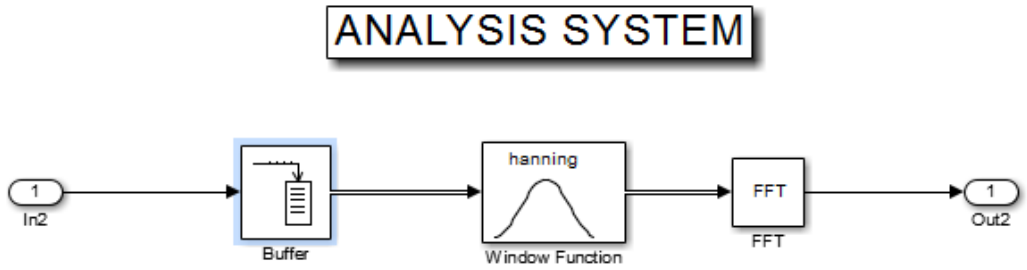

Figura 117: Subsistema del proceso de análisis del Vocoder de fase.

El resultado de la FFT entrega a su salida señales complejas con valores de módulo y fase, por lo que son separadas mediante el bloque *Complex to magnitude-angle*. La figura [118] muestra el subsistema de procesado intermedio entre los procesos de análisis y síntesis. Es en este bloque donde pueden añadirse los procesados necesarios sobre cada fragmento de la *STFT* para conseguir el efecto de audio deseado.

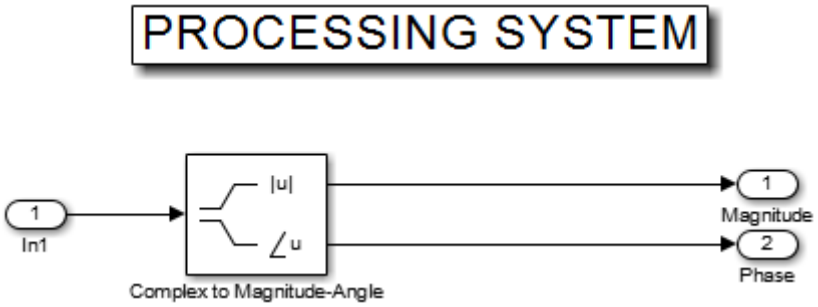

Figura 118: Subsistema de procesado de la STFT.

Finalmente se presenta en la figura [119] el bloque de síntesis, que contiene el proceso de cálculo de la fase para la reconstrucción descrito anteriormente junto con los procesos de FFT inversa, enventanado y aplicación del método *Overlap-Add*. El bloque *Principal Argument* contiene la estructura de cálculo correspondiente a la ecuación [86], y se muestra en la figura [120]. El bloque *Magnitude-Angle to Complex* permite volver a juntar el módulo y la fase de la FFT en un número complejo para así poder calcular la FFT inversa. Al final del sistema se realiza el proceso de atenuación de la señal de salida descrito en la ecuación [92].

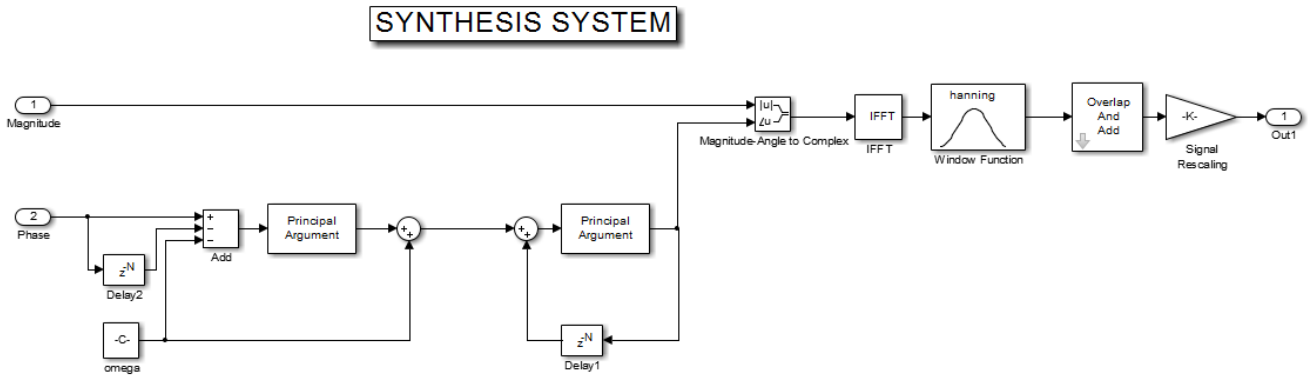

Figura 119: Subsistema de síntesis del Vocoder de fase.

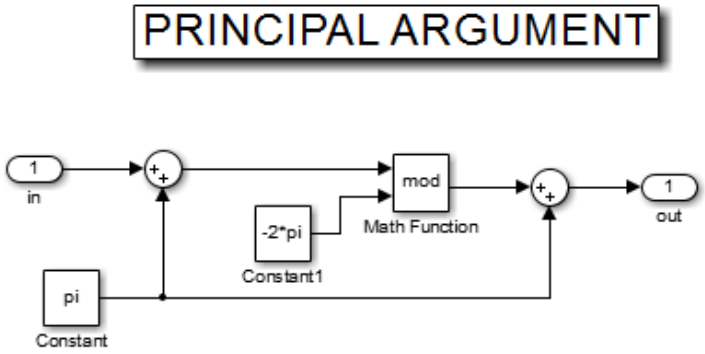

Figura 120: Subsistema correspondiente a la función principal argument.

La figura [121] [ref. 17] muestra el proceso de *Overlap-Add*, donde la señal de entrada es dividida en fragmentos de N-L y L, siendo N el tamaño de la ventana. Cumpliendo con el método, el fragmento N-L empezando por el comienzo de la ventana es concatenado con el fragmento N-L de la ventana anterior, empezando por el final de esta ventana. A continuación se guarda el fragmento N-L para la siguiente iteración, y se sacan las L primeras muestras tras la concatenación.

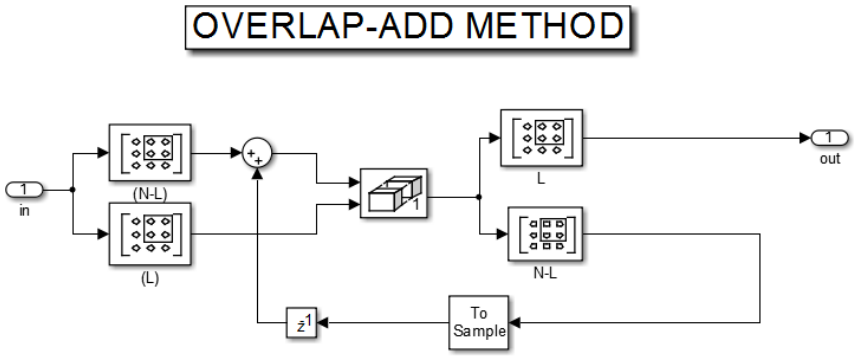

Figura 121: Subsistema del método Overlap-Add

A continuación se presenta la señal sinusoidal tras pasar por el *Vocoder* de fase, configurado con un tamaño de ventana de 1024 muestras e intervalos de solape iguales para el proceso de análisis y de síntesis de 256 muestras. La gráfica muestra la señal perfectamente reconstruida tanto en amplitud como en fase.

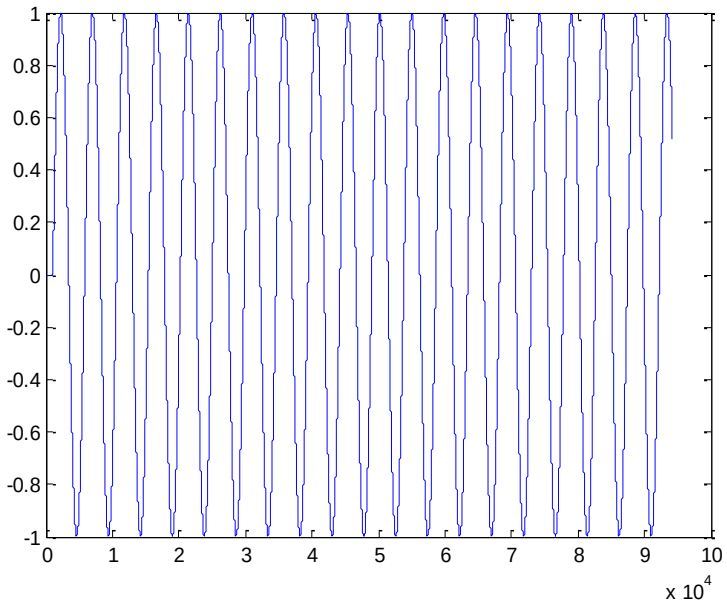

Figura 122: Señal sinusoidal de 10Hz reconstruída tras pasar por el Vocoder de fase. Window Length = 1024 muestras; Analysis hop size= 256 muestras = synthesis hop size.

7.4. TIME STRETCHING MEDIANTE EL VOCODER DE FASE

Desde la interpretación mediante *STFTs* del *Vocoder* de fase puede entenderse cómo conseguir la compresión o expansión temporal de la señal de audio sin modificar el *pitch*. Al seleccionar intervalos de solape de la misma dimensión, la señal de salida se corresponde perfectamente en la escala temporal con la señal de entrada. Modificando alguno de estos intervalos de solape produce que la señal de salida tras pasar por el *Vocoder* sea de una duración diferente, aumentando o disminuyendo su longitud en un factor $\alpha = \frac{synthesis\ hop}{analysis\ hop}$.

Este efecto se debe a que al tomar intervalos de solape distintos, los procesos de FFT y FFT inversa toman segmentos espaciados de forma diferente en el tiempo, produciendo que los cambios en frecuencia sean más rápidos o más lentos en el proceso de síntesis. Si el intervalo de solape en el proceso de síntesis es mayor que el intervalo de solape en el proceso de análisis, la señal es expandida en el tiempo. En el caso contrario, la señal sufre una compresión en el tiempo.

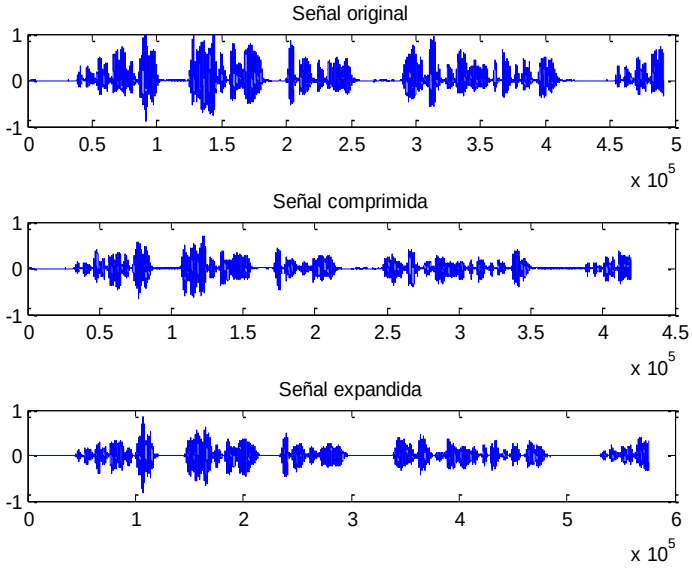

Figura 123: Time Stretching de la señal de voz „speech.wav". Window Length: 1024 muestras. Señal Comprimida: Analysis hop size: 300 muestras; Synthesis hop size: 256 muestras. Señal Expandida: Analysis hop size: 256 muestras; Synthesis hop size: 300 muestras.

Al realizar este proceso de compresión o expansión, la fase de la señal tras el proceso de análisis ya no coincide en el tiempo con la fase de la señal correspondiente para su reconstrucción. Es necesario multiplicar la fase por el mismo factor de compresión o expansión α, tal y como muestra la figura [124], de forma que este efecto sea corregido, ya que produce artificios en frecuencia. Como ejemplo, si por cada intervalo de tiempo la fase aumenta 45º, al producir una expansión con un factor α = 2, el incremento de fase de 45º se corresponde con un intervalo de tiempo del doble, por lo que el incremento de fase por cada intervalo de tiempo debería ser 90º.

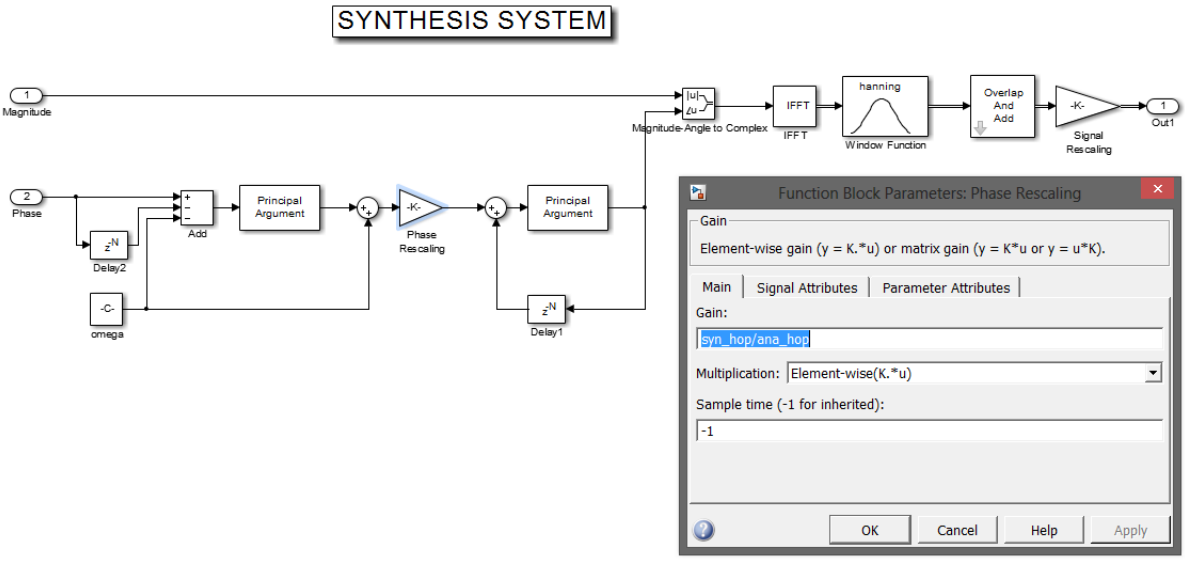

Figura 124: Vocoder de fase configurado para el proceso de Time Stretching.

7.5. PITCH SHIFTING MEDIANTE EL VOCODER DE FASE

Para modificar el *pitch* de una señal de audio mediante un *Vocoder* de fase se sigue la misma dinámica establecida en el apartado [6.4]. La señal sufre un proceso de remuestreo o resampling según un factor $\frac{ana\_hop}{syn\_hop}$, compensando previamente o a posteriori el cambio de la escala temporal mediante el proceso de *Time stretching* descrito anteriormente, función realizada por el propio *Vocoder*.

Este proceso también puede realizarse cambiando la frecuencia de muestreo en el conversor D/A según el factor que se desee. Sin embargo, se recomienda utilizar el proceso de *resampling* para que así la frecuencia de muestreo sea la misma, tanto en la entrada como en la salida.

El proceso de *resampling* se ha descrito en el apartado [1.5]. La señal pasa por el proceso de interpolación según el factor *analysis hop size*. Seguidamente es filtrada mediante un filtro FIR de ganancia el factor de interpolación y frecuencia de corte según la expresión [93]. A continuación se realiza el proceso de diezmado según el factor *synthesis hop size.*

$$\omega_c = min\left[\frac{\pi}{L}, \frac{\pi}{M}\right] \quad (93)$$

Aplicar el proceso de *resampling* en Simulink es sencillo gracias al bloque *FIR Rate Conversion.* Este bloque permite realizar los tres procesos anteriores, seleccionando el factor de interpolación, diezmado, y los coeficientes del filtro FIR. Estos coeficientes se han obtenido mediante la función de Matlab *fir1(N-1, ω_c).* La figura [125] muestra la ventana de configuración del bloque. Se recomienda utilizar un orden del filtro mayor al factor de interpolación para así disponer de suficiente resolución espectral.

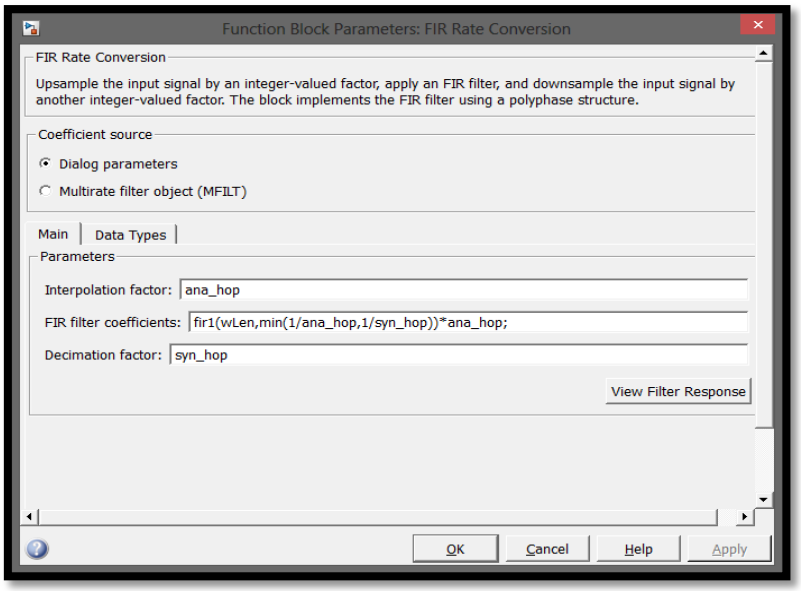

Figura 125: Ventana de configuración del bloque FIR Rate Conversion.

La respuesta en frecuencia del filtro FIR se muestra en la figura [126], con una configuración de ganancia analysis hop size de 256 (48,2 dB), frecuencia de corte ω_c = 1/256 y orden del filtro 1024.

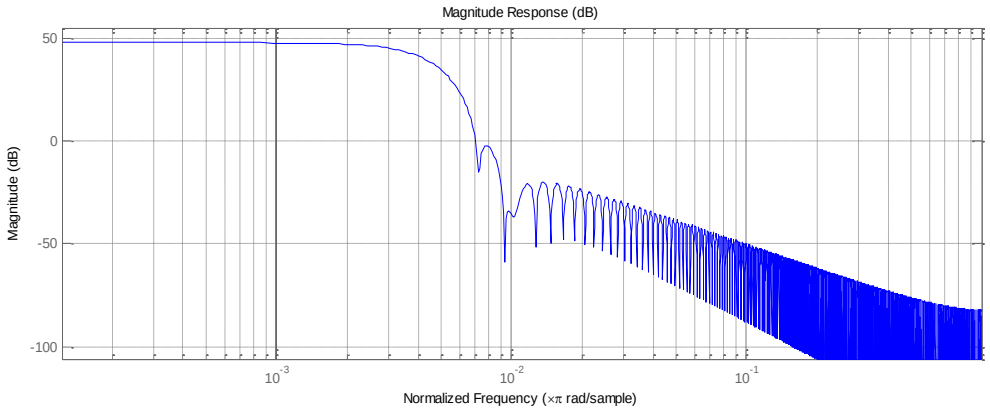

Figura 126: Respuesta en frecuencia del filtro FIR. Ganancia: 48,2 dB. Frecuencia de corte ω_c = 1/256.Orden: 1024.

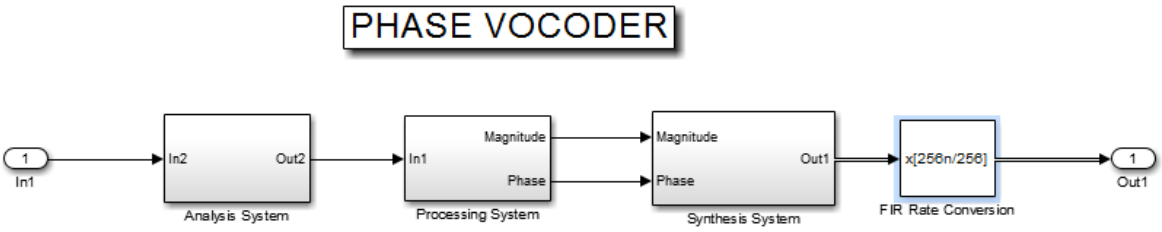

Figura 127: Vocoder de fase para el proceso de Pitch Shifting.

7.6. CONCLUSIÓN

El procesado en el dominio del tiempo-frecuencia puede dar lugar a una serie de efectos de distinta índole muy útiles para los procesos de audio, como son filtrados variantes en el tiempo, filtrados no lineales, mezclas de distintos timbres o sonidos, efecto de robotización, *denoising*, etc…

En este proyecto el objetivo del *Vocoder* ha sido la compresión y expansión temporal. Comparando con los procesos del capítulo anterior, el *Vocoder* de fase produce un efecto de mucha mayor calidad y fiabilidad, siendo posible su ámbito de uso con distintos tipos de señales, ya sean señales de voz, señales musicales, etc… El proceso llevado a cabo permite comprimir y expandir en el tiempo evitando clips indeseados y artificios, para un amplio margen del factor de compresión/expansión. Ocurre lo mismo con los cambios de *pitch*, dando lugar a un sonido final modificado en frecuencia prácticamente libre de artificios.

En este capítulo, Simulink se comporta como una herramienta perfecta para el desarrollo del *Vocoder* de forma rápida y sencilla. Los bloques de enventanado y *Buffers* permiten realizar el proceso de análisis y síntesis rápidamente. La facilidad de Simulink para realizar también transformadas FFT es un gran aliciente para su uso en este tipo de procesados. Tanto en este algoritmo como en el ecualizador gráfico *FFT*, el cual utiliza la misma técnica de *STFT,* se han utilizado las **máscaras** en subsistemas permitiendo configurar de forma rápida los parámetros que rigen el comportamiento del *Vocoder*.

Finalmente, el modelo propuesto, con un sistema de codificación mucho más avanzado como C++ puede permitir el uso del *Vocoder* en tiempo real con baja latencia. Simulink presenta un código más lento por lo que se acumula cierto retardo a la hora de escuchar la señal de salida. Esta es otra línea de desarrollo e investigación que puede llevarse a cabo con una gran cantidad de efectos existentes en la actualidad.

8. IMPLEMENTACIÓN DE PLUGINS

8.1. INTRODUCCIÓN

El objetivo final de todo algoritmo de procesado de audio dentro del ámbito descrito es llevar a cabo una aplicación de una determinada utilidad, configurable por el usuario mediante una serie de parámetros sin necesidad de conocer su funcionamiento interno. Dependiendo del ámbito de aplicación del procesado, debe tenerse en cuenta que el propio usuario dispondrá de conocimientos mínimos sobre el tema, por lo que un buen producto final será exitoso gracias a su funcionabilidad estrechamente ligada a su facilidad de uso.

En el mundo de la ingeniería de audio, en la mayor parte de las ocasiones las aplicaciones finales serán utilizadas por técnicos de sonido en distintos ambientes como estudios de grabación o post-producción, situaciones de directos musicales, etc… El elemento común a todas estas situaciones es la herramienta general con la que se trabaja, las denominadas **estaciones de trabajo de audio** o ***DAW*** *(Digital Audio Workstation).* Existen estaciones de trabajo de diversa índole, tanto en *hardware* como en *software*, las cuales disponen prácticamente en su totalidad de un *software* de edición de audio, donde se suele integrar todo el control y funcionamiento de la estación junto a una mesa de mezclas. Algunos ejemplos de estos programas son los extendidos *Cubase/Nuendo* o *Pro Tools*, aunque existe una gran infinidad de programas de grandes prestaciones similares a los anteriores.

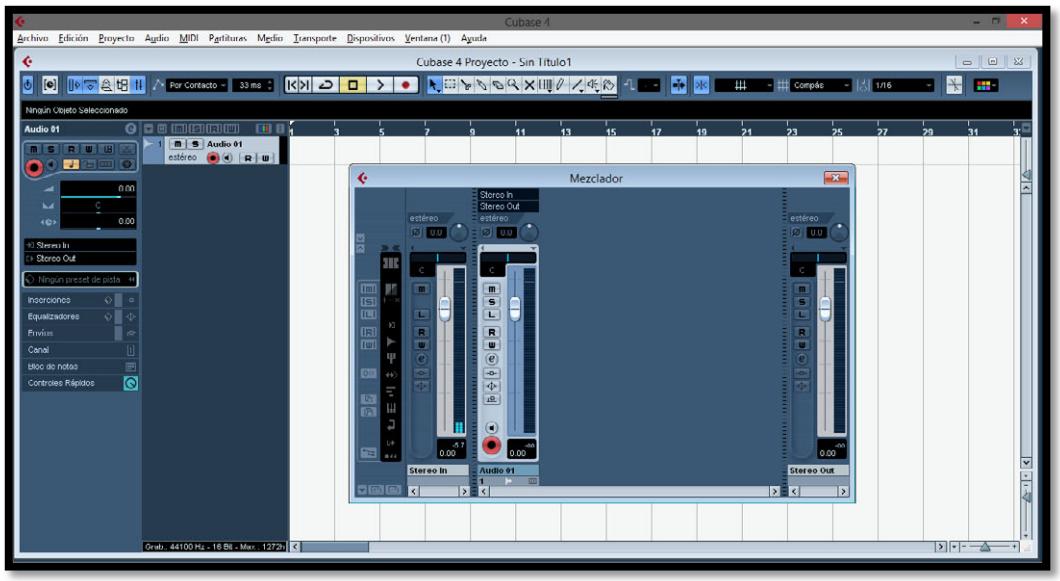

Figura 128: Entorno de trabajo del software Cubase Studio, propiedad de la compañía STEINBERG.

Dentro de todos estos programas existen aplicaciones que ejecutan determinados procesados o efectos y que pueden estar desarrollados por otros individuos o compañías. Son los denominados *plugins* de audio. La forma de poder implementar estas aplicaciones dentro de un *software* de audio fue ideada por la compañía STEINBERG, creando una interfaz común para estas aplicaciones denominada ***VST*** *(Virtual Studio Technology)*. De esta forma, cualquier programa desarrollado bajo la tecnología *VST* puede implementarse dentro de un *software* de audio compatible, denominados ***VST Host***, y así incluir su proceso en la cadena de una estación de trabajo digital.

Llegado a este punto, el objetivo final de este proyecto es establecer un método sencillo para poder establecer el puente entre los algoritmos desarrollados en Simulink y su uso práctico en un *software* de audio. Construir este puente no es una tarea sencilla, ya que se requiere de conocimientos acerca del procesado de audio en lenguajes como C o C++, además del uso de las librerías VST propiedad de STEINBERG. Además, existen varias cuestiones a tener en cuenta a la hora de desarrollar estos algoritmos en sendos lenguajes. Es necesario un control de la latencia de la aplicación, aspecto que puede arruinar su funcionabilidad. Por otro lado, es necesario programar una interfaz gráfica para el control del *plugin*, la cual debe disponer también de una estética amena para el usuario. Y por último, es necesario conocer a la perfección el algoritmo a aplicar, habiendo probado antes su funcionamiento, función en la cual encaja perfectamente el método de prototipos mediante Simulink.

Sigue quedando en pie la pregunta, ¿cómo realizar de forma rápida y sencilla el puente entre Simulink y los *plugins VST* para poder dar forma a una aplicación final? La respuesta ha sido lograda por la empresa DLAB GMBH, la cual ha desarrollado un programa llamado ***AUDIO PLUGIN GENERATOR***, que permite pasar los modelos de Simulink a formato *plugin VST*.

8.2. AUDIO PLUGIN GENERATOR

Audio Plugin Generator es un programa capaz de establecer el puente entre Simulink y el *software* de audio de una estación de trabajo digital. Este programa permite compilar los modelos de Simulink en código C, para así enlazar con las librerías *VST* y poder generar un *plugin* con extensión *„dll"*. El archivo de *plugin* finalmente puede ser incluido en un *VST Host* o *software* de audio. Este *plugin* es configurado mediante un archivo de interfaz de gráfica de usuario (en inglés *UIC: User Interface Configuration file*). La figura [129] muestra el diagrama de bloques del proceso llevado a cabo por el programa.

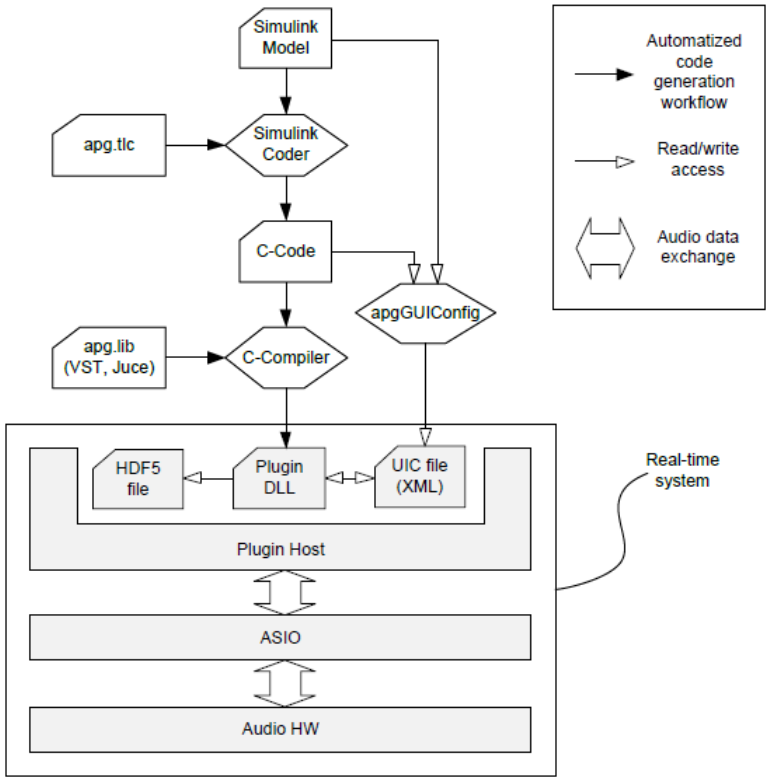

Figura 129: Diagrama de bloques del proceso llevado a cabo por Audio Plugin Generator.

Audio Plugin Generator presenta algunos requisitos recogidos en su guía de usuario [ref. 15]. En primer lugar, sólo es compatible con las versiones de Matlab R2009a o superior. Seguidamente, es necesario disponer de un compilador compatible en Matlab. Se recomienda utilizar el compilador escogido por este proyecto (año 2014), ***Windows SDK 7.1*** al ser el compilador actual con mayor compatibilidad con los módulos de Matlab/Simulink. En el anexo [A] se describe detalladamente el proceso de instalación para su uso en Matlab.

La versión gratuita de *Audio Plugin Generator* tiene una limitación por la cual los modelos de Simulink sólo permiten un canal de entrada y de salida, suficiente para los algoritmos desarrollados en este proyecto. Por tanto esta versión no puede procesar señales estéreo. En caso de necesitar procesados multicanal, es necesario utilizar la versión completa del programa. El programa presenta una librería en Simulink (*APG blockset)* con los bloques de señal de entrada, señal de salida y visor de la señal (*Access Point*), no configurables, para establecer la entrada y salida del algoritmo y los puntos de visualización de la señal. Estos bloques se muestran en la figura [130].

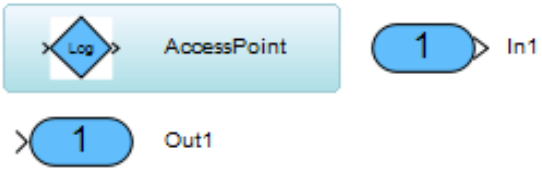

Figura 130: Librería APG blockset en Simulink.

8.2.1. COMPILACIÓN DEL MODELO EN SIMULINK

Una vez finalizado el algoritmo en un modelo de Simulink, debe configurarse el compilador para poder pasar el modelo a un *plugin „.dll"*Es importante tener en cuenta la frecuencia de muestreo con la que se trabaja. La frecuencia de muestreo seleccionada en Simulink es independiente de la seleccionada en el *software* de audio, por lo que es responsabilidad del propio usuario del *software* de audio seleccionar la frecuencia de muestreo correcta del proyecto sobre el que se esté trabajando.

Una vez establecida la frecuencia de muestreo adecuada en Simulink, deben estar seleccionadas en la ventana de *Configuration Parameters/Solver* las opciones ***Solver: Fixed-step; Discrete***, ya que los *plugins VST* trabajan con señales discretas de una frecuencia fija de muestreo. La figura [131] muestra la ventana de configuración con estas características.

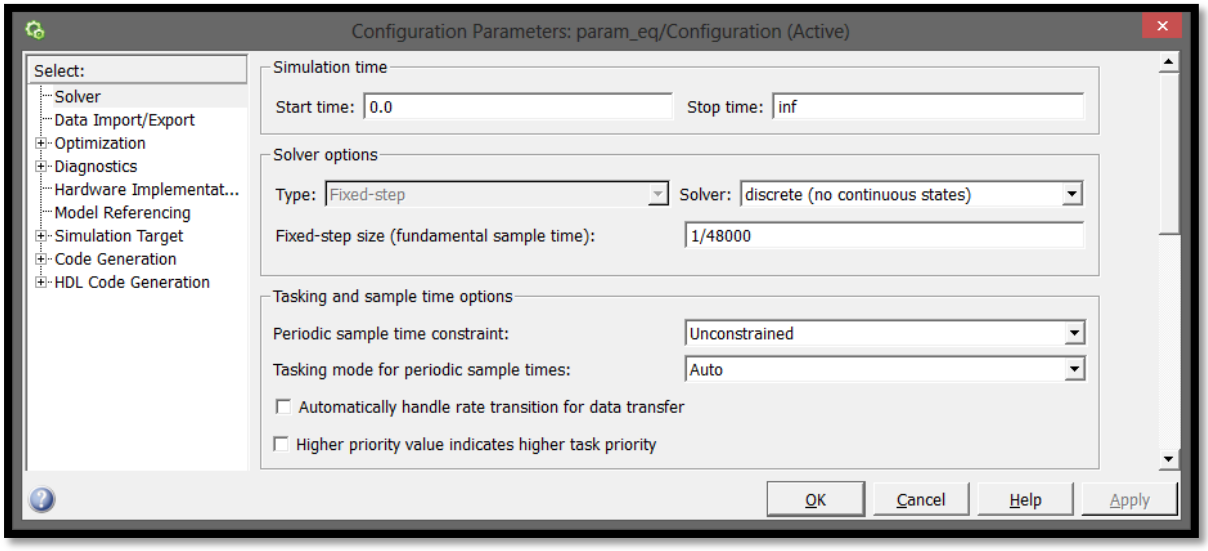

Figura 131: Ventana de configuración de los parámetros del modelo en Simulink.

A continuación debe configurarse el módulo de compilación de Simulink accediendo a la sección *Code Generation*. Debe seleccionarse las librerías „***apg.tlc***" en la opción *System Target file.* En la subsección de *Interface,* en *Code generation*, debe comprobarse que la casilla „***Classic call Interface***" esté seleccionada. Las figuras [132] y [133] muestran la ventana de configuración de ambos parámetros.

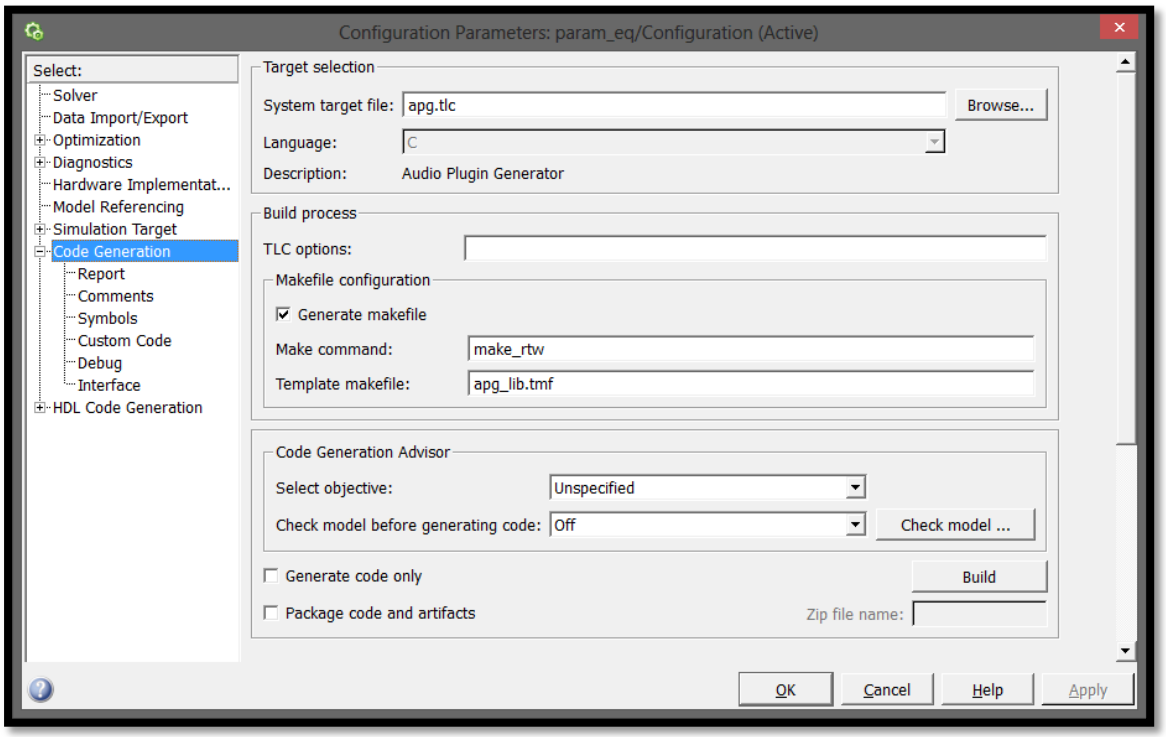

Figura 132: Ventana de configuración de la sección Code Generation.

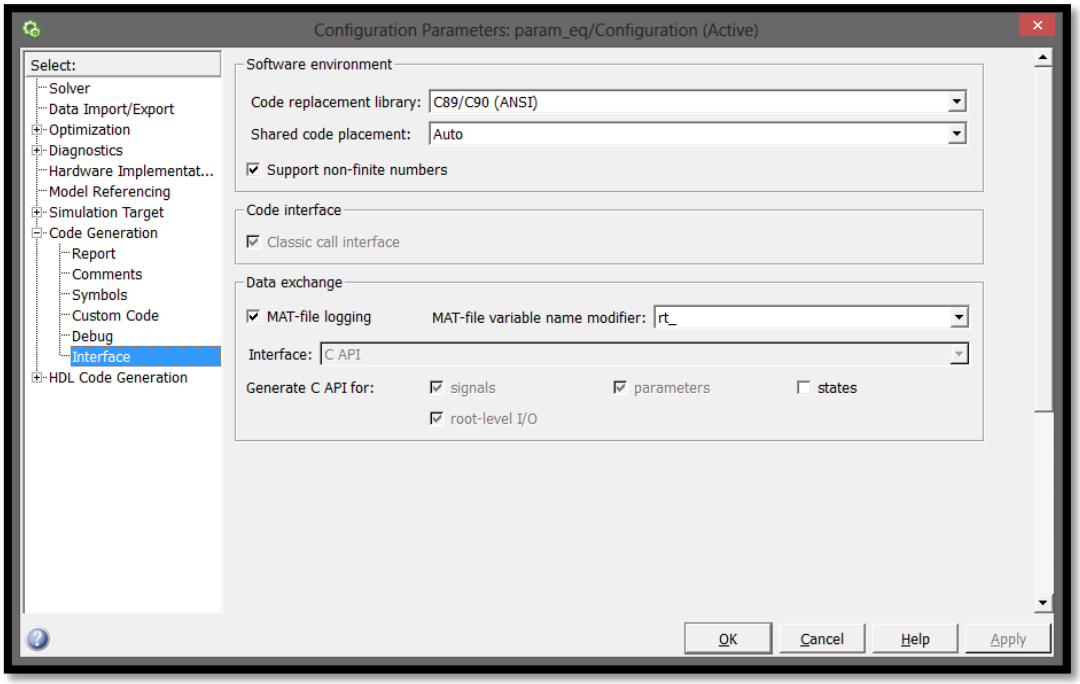

Figura 133: Ventana de configuración de la subsección Interface, dentro de Code Generation.

Una vez establecida la configuración, puede procederse a la compilación del modelo en Simulink, presionando el botón *„**Build Model**"*, en la ventana principal del modelo o en la ventana de configuración de parámetros, en la sección *Code Generation.* Este proceso generará un archivo *„dll"* en el subdirectorio del proyecto *„work/apg/"*.

8.2.2. CONFIGURACIÓN DE UNA INTERFAZ GRÁFICA DE USUARIO (GUI)

Una vez obtenido el archivo *„dll"*, debe realizarse la configuración del archivo correspondiente a la interfaz gráfica de usuario, en extensión *„uic"*. Para ello, el programa *APG* incluye una función en Matlab que permite seleccionar las variables del modelo que se corresponderán con los potenciómetros configurables, así como la descripción del *plugin*, el tamaño de la ventana de configuración y las visualizaciones de la señal, entre otras opciones.

Con el modelo correspondiente abierto en Simulink y el directorio de Matlab establecido en la subcarpeta del proyecto *„work"*, debe ejecutarse en la ventana de comandos el archivo *„**apgGUIConfig.m**"* (Comando *„open apgGUIConfig.m"*). De esta forma se abrirá una ventana para la configuración de los parámetros del *plugin*, tal y como muestra la figura [134]. En caso de estar utilizando la versión de Matlab aún con el menú *„Start"* puede ejecutarse la aplicación accediendo a la sección *„Toolboxes/Audio Plugin Generator/GUI Configuration"*.

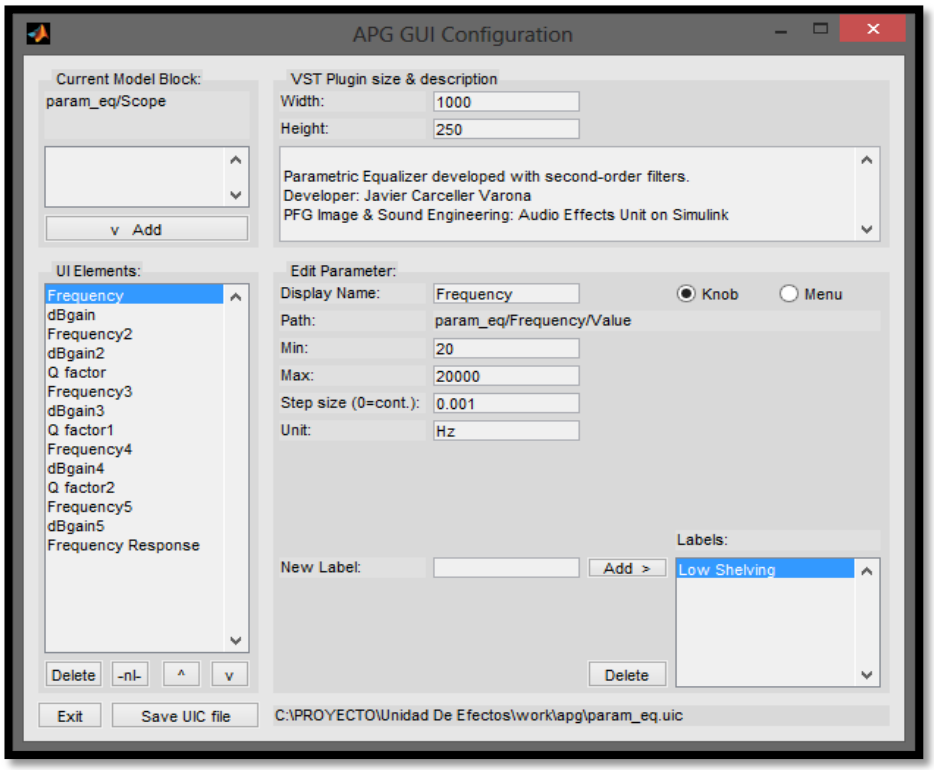

Figura 134: Ventana de configuración de la interfaz gráfica de usuario mediante la función 'apgGUIConfig.'m.

Entre las distintas opciones presentes en la ventana de configuración, las opciones *„Width"* y *„Height"* permiten seleccionar la anchura y la altura de la ventana del *plugin.* Seguidamente puede establecerse una descripción del *plugin* en el espacio a continuación. Para establecer parámetros del modelo como potenciómetros configurables, debe seleccionarse en primer lugar el bloque deseado en el modelo en Simulink. Existen determinados bloques cuyos parámetros pueden ser establecidos como potenciómetros, pero no todos ellos son compatibles. Una vez seleccionado el bloque, aparecerá en el cuadro de la esquina superior izquierda, tal y como muestra la figura [135]. Dependiendo del bloque seleccionado aparecerá una o varias opciones a agregar como potenciómetros.

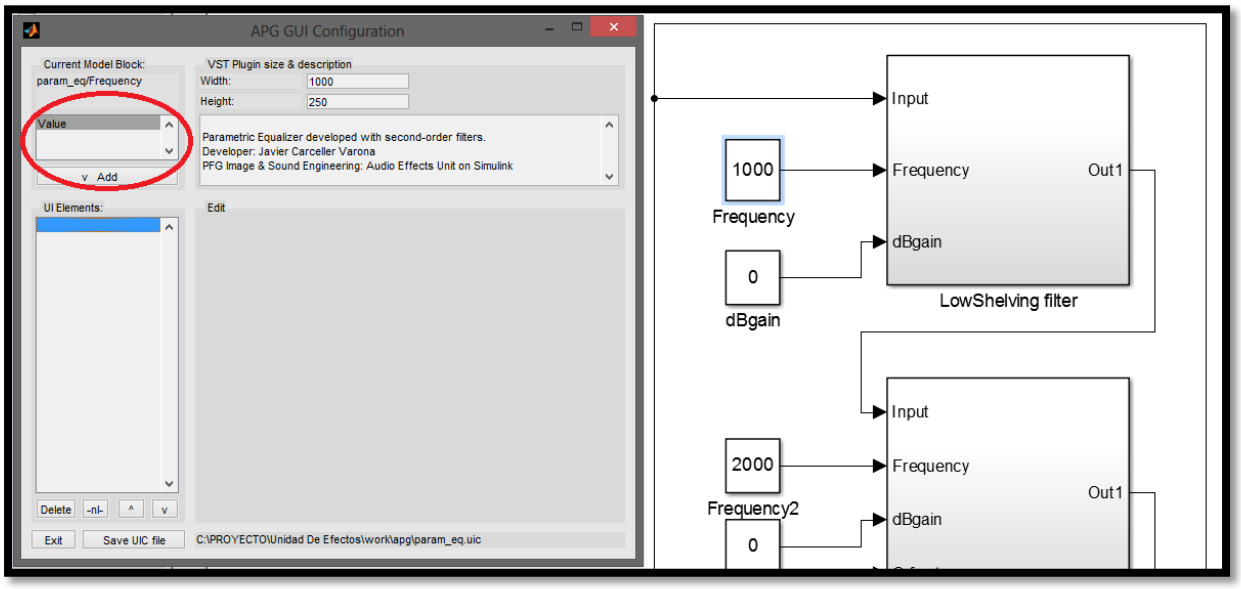

Figura 135: Selección de un bloque del modelo en Simulink como potenciómetro.

Una vez seleccionado, pulsando el botón *„Add"* el valor determinado pasa a ser un potenciómetro configurable. Seleccionando entre *„knob"* o *„menu"* aparecerá su visualización como un potenciómetro o como un menú. Los parámetros configurables son el nombre con el que se muestra el potenciómetro (*„display"*), el valor mínimo y máximo, el tamaño del escalón (*„step size"*), y las unidades (*„unit"*). Pueden mostrarse también etiquetas que permitan diferenciar unos potenciómetros de otros en la opción *„Labels"*. Una vez establecidos todos los parámetros, pulsando la opción *„Save UIC file"* se guardará el archivo en el mismo directorio donde se encuentra el archivo *„.dll".* Para utilizar los *plugins*, ambos archivos deben ser copiados en el mismo directorio, en la carpeta correspondiente del *software* de audio determinado.

La figura [136] muestra la ventana de configuración del *plugin* correspondiente al ecualizador paramétrico, junto con el visualizador de la respuesta en frecuencia. Se incluye en el anexo [B] una descripción detallada de los parámetros para la configuración de los *plugins* desarrollados en este proyecto. Los *plugins* han sido probados en el *software* de audio *Reaper v.4611,* sobre el sistema operativo Windows 7.

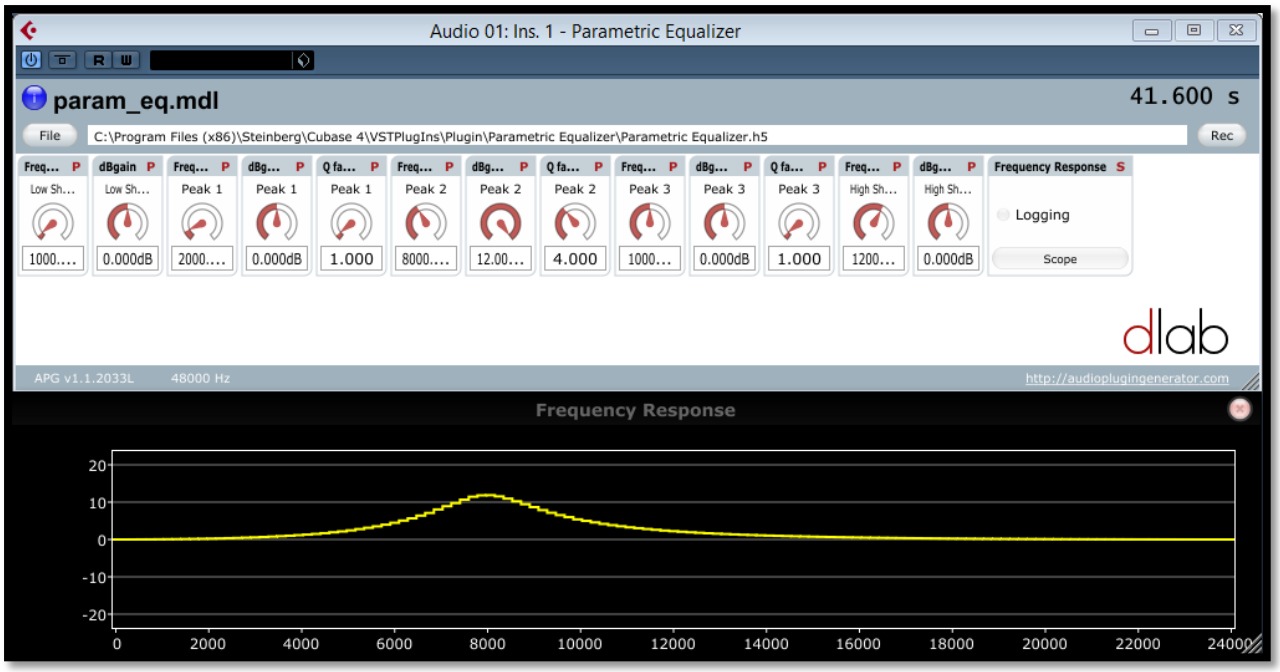

Figura 136: Ventana de configuración del plugin parametric EQ.

8.2.3. GRABACIÓN DE DATOS

Una de las novedades presentes en los *plugins* desarrollados con *APG* es la capacidad de grabación y registro de señales registradas en el *plugin* mediante el bloque *„Access Point"*. Seleccionando en el *plugin* la opción *„Rec"* y el directorio de destino, la señal deseada es registrada en formato **HDF5**.

HDF5 es un formato desarrollado como *software* libre para el registro de señales de gran tamaño, compatible con Matlab junto con una gran cantidad de aplicaciones distintas. Este sistema de registro se convierte en una herramienta de gran utilidad a la hora de realizar la prueba final de prototipos en un *software* de audio, pudiendo registrar determinadas señales para su posterior estudio y detección de errores del *plugin*.

Para poder cargar un archivo HDF5 es necesario utilizar la función *„apgLogRead.m"* presente en la librería de *APG*. Esta función tiene un comportamiento similar al de la función para cargar archivos de audio *„wavread"*, entregando como salida una matriz correspondiente a la señal registrada [ref. 15].

9. CONCLUSIONES

En este proyecto se han abordado las capacidades de una herramienta cuyo uso está aumentando considerablemente en el ámbito profesional de la ingeniería. Simulink es a grandes rasgos una herramienta muy potente capaz de trabajar en distintas áreas y de simplificar el proceso de programación, ahorrando por tanto en costes y tiempo. Sus capacidades van aún más allá de los algoritmos desarrollados en este proyecto, y sus constantes actualizaciones y nuevas librerías agrandan aún más sus virtudes.

Uno de los aspectos más importantes obtenidos trabajando con Simulink es su capacidad como herramienta didáctica. El uso de Matlab está muy extendido en las universidades de ingeniería. Sin embargo, la programación en Matlab en ocasiones puede suponer un impedimento y un gasto de tiempo no deseado en el proceso docente. Durante la búsqueda de información en este proyecto, se han encontrado una gran cantidad de referencias bibliográficas de profesores que han implementado Simulink como herramienta en la enseñanza. La principal razón es su interfaz de programación en diagramas de bloques, diagramas muy utilizados por los ingenieros como concepto de desarrollo. Este proyecto propone Simulink como una herramienta a utilizar en el proceso docente actual, no solo por sus capacidades si no por su mayor simplicidad de uso.

Gracias al diseño de algoritmos de procesado digital y a la búsqueda de información, se ha logrado el objetivo de profundizar en los sistemas actuales de procesado digital de la señal. Se han abordado cuatro áreas fundamentales del procesado de audio, con diferentes resultados en cada una de ellas. Es en el procesado de frecuencia donde se ha encontrado una mayor complejidad en cuanto al algoritmo a desarrollar. Mientras que la construcción de un ecualizador paramétrico es sencilla, los ecualizadores gráficos perfectos aún están en proceso de desarrollo. Sin embargo, Simulink cumple con las expectativas con una gran librería preparada para filtrar señales.

Las estructuras en bloques permiten crear de forma rápida algoritmos de procesado de dinámica y retardos, con bloques preparados para simplificar estos procesos. De la misma forma sucede con el caso del *Vocoder* gracias a los bloques preparados para el proceso de análisis-síntesis. Sin embargo a la hora de realizar los procesos de compresión y expansión temporal se presenta una dificultad, el algoritmo implementado en Matlab no es compatible con Simulink, ya que se trabaja con una señal cuya dimensión de salida se desconoce. Además, establece una norma fundamental para la programación en Simulink: es necesario que el algoritmo funcione como un flujo de datos.

Por otro lado, la herramienta *APG* presenta una buena idea a la hora de diseñar prototipos de algoritmos digitales. Sin embargo, esta herramienta tiene actualmente una serie de limitaciones y necesita ser desarrollada más a fondo. Los *plugins* construidos en lenguaje C por *APG* pueden ser inestables en algún *software* de edición de audio. Existen limitaciones con respecto a la versión de Windows de 64 bits, además de una instalación complicada debido a incompatibilidades con los *SDK* de Windows. Por otro lado, la falta de configuración de una interfaz gráfica con un diseño propio lo convierten en una herramienta útil únicamente para diseñar prototipos, y el resultado final no es un producto comercializable.

Los *plugins* desarrollados funcionan correctamente con señales musicales y de voz. Por tanto, el sistema de diseño de algoritmos de audio propuesto por este proyecto es viable para el proceso de diseño de prototipos. Se esperan mejoras en el *software* de *APG* que permitan una mayor fiabilidad, con más herramientas de control para comprobar el funcionamiento de los algoritmos, y una mayor estabilidad. Para el diseño de *plugins* definitivos, se recomienda el uso de lenguajes de programación C o C++ una vez se ha diseñado el algoritmo en Simulink.

Como última reflexión, la investigación en el mundo del audio no ha terminado. Los sistemas de audio deben adaptarse a los gustos de los intérpretes y músicos, y estos gustos están en constante cambio. Por ello, son y serán necesarios nuevos algoritmos que se adapten a sus necesidades. Existen aún algoritmos imperfectos que deben ser perfeccionados, y siguen surgiendo nuevas técnicas de procesado digital que pueden dar lugar a nuevos descubrimientos. Además, este ámbito no sólo cubre el uso de algoritmos en el mundo musical, sino también en otras áreas como la biomedicina y el tratamiento de la voz o el tratamiento de señales mediante plataformas como Arduino, donde Simulink puede aportar un papel fundamental.

ANEXOS

ANEXO A: PASOS PARA LA INSTALACIÓN DE WINDOWS SDK 7.1

Este anexo describe los pasos necesarios para instalar la herramienta de compilación actual utilizada por Matlab/Simulink para transformar archivos de ambos programas en código C. Los pasos han sido llevados a cabo en Windows 7 32 bits.

a) **Descargas:**

- Compiladores compatibles con Matlab Simulink para Windows de 32 bits. Seleccionar en el enlace Windows SDK 7.1: http://www.mathworks.es/support/compilers/R2012a/win32.html.
- Compiladores compatibles con Matlab Simulink para Windows de 64 bits. Seleccionar en el enlace Windows SDK 7.1: http://www.mathworks.es/support/compilers/R2012a/win64.html.
- Microsoft Net Framework 4.0: http://www.microsoft.com/en-us/download/details.aspx?id=17851.
- Actualización de Visual C++ 2010 Express: http://www.microsoft.com/en-us/download/details.aspx?id=4422.

b) **Pasos previos:**

- Desinstalar *Microsoft Visual C++ 2010 Redistributable* del equipo (Produce un error al instalar *Windows SDK 7.1*). Será reemplazado por una versión diferente al instalar *Windows SDK 7.1*.
- Instalar Net Framework 4.0, necesario para el correcto funcionamiento de *Windows SDK 7.1*.

c) **Instalación:**

- Ejecutar el archivo de instalación de *Windows SDK 7.1*.
- Seleccionar todas las opciones mostradas en la figura [A.1].
- Esperar hasta que termine la descarga e instalación para finalizar.
- Instalar la actualización para *Windows SDK 7.1* con los compiladores Visual C++ 2010 Express.

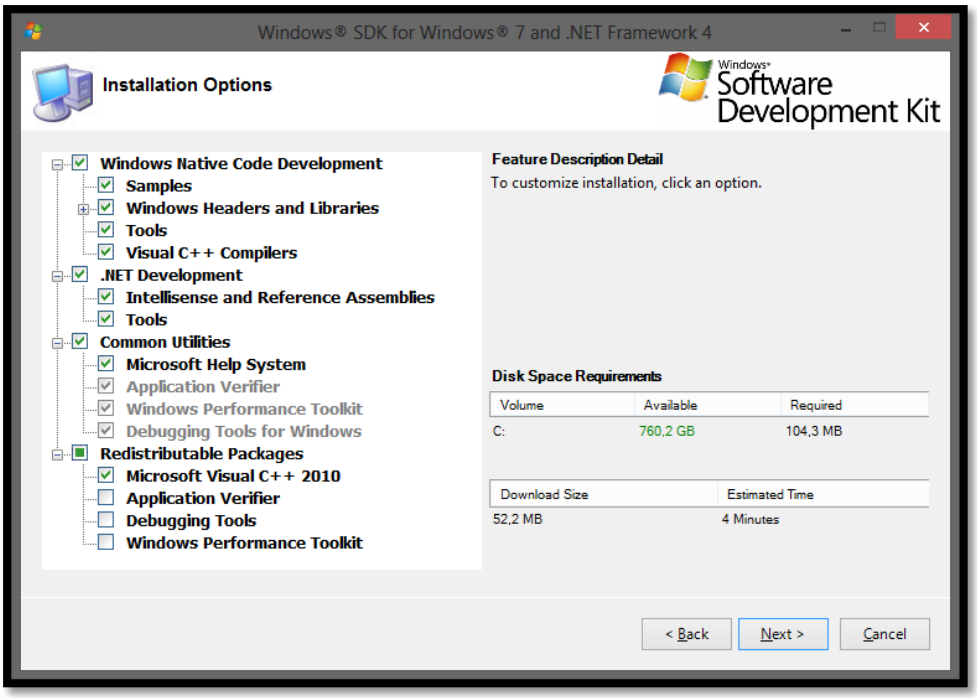

Figura A. 1: Ventana de instalación del paquete Windows SDK 7.1.

d) **Configuración en Matlab:**

- Escribir en la ventana de comandos „***mex –setup***".
- Escribir „*y*"para permitir a Matlab mostrar la lista de compiladores instalados.
- Seleccionar el compilador *Windows SDK 7.1*, tal y como muestra la figura [A.2].

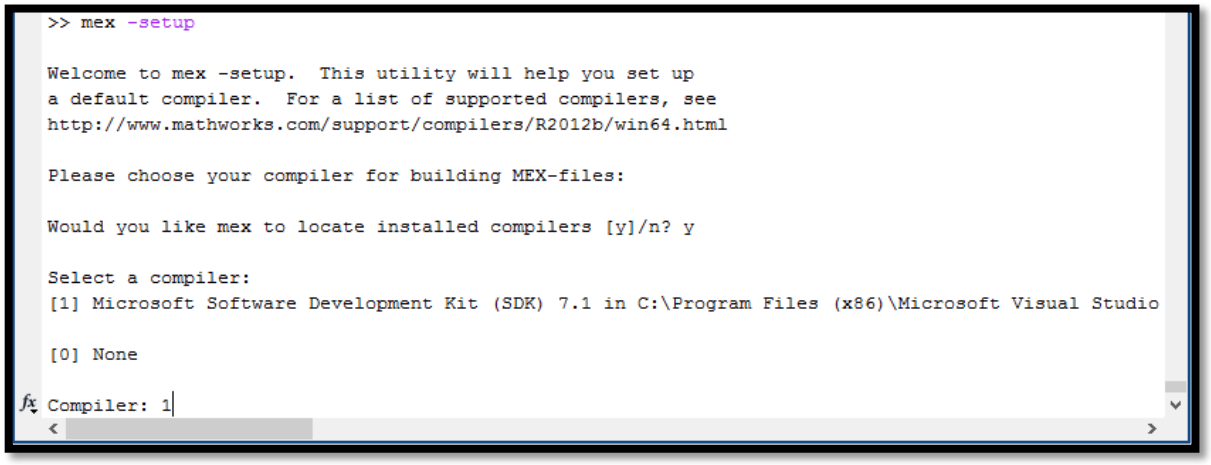

Figura A. 2: Ventana de comandos de Matlab para configurar el compilador a utilizar.

ANEXO B: MANUAL DE USO DE AUDIO EFFECTS UNIT

En este anexo se describen todas las opciones de configuración de los *plugins* presentes en *Audio Effects Unit.* Esta unidad de efectos está formada por los *plugins* desarrollados en este proyecto, agrupados según su topología. Se describen el funcionamiento y configuración de cada uno de los parámetros presentes.

Para instalar el paquete de *plugins* en el *software* de audio, debe copiarse la carpeta *Audio Effects Unit* en el directorio de *plugins VST* del programa (en el caso de *Reaper*, el directorio destino es „*Reaper/Plugins/FX/*"). Todos los *plugins* trabajan a una frecuencia de muestreo fija de 48kHz. Debe configurarse el proyecto en el *software* de audio para trabajar a esta frecuencia.

B.1. EQUALIZERS

a) **Parametric Equalizer:**

- Ecualizador *Low Shelving:*
  - *Frequency* (Hz): Control de la frecuencia de corte a -3dB. Rango: [20Hz-20kHz].
  - *dBgain* (dB): Control de ganancia del ecualizador en la banda de paso. Rango [-12dB +12dB].

- Ecualizadores *Peak 1, 2 y 3:*
  - *Frequency* (Hz): Control de la frecuencia de corte a -3dB. Rango: [20Hz-20kHz].
  - *dBgain* (dB): Control de ganancia del ecualizador en la banda de paso. Rango [-12dB +12dB].
  - *Q factor*: Control del factor de calidad del ecualizador *Peak.* Rango: [0.5-10].

- Ecualizador *High Shelving:*
  - *Frequency* (Hz): Control de la frecuencia de corte a -3dB. Rango: [20Hz-20kHz].
  - *dBgain* (dB): Control de ganancia del ecualizador en la banda de paso. Rango [-12dB +12dB].
- *Frequency Response:* Visualiza la respuesta en frecuencia de la señal en tiempo real, mediante ruido blanco. Rango [0-24kHz].

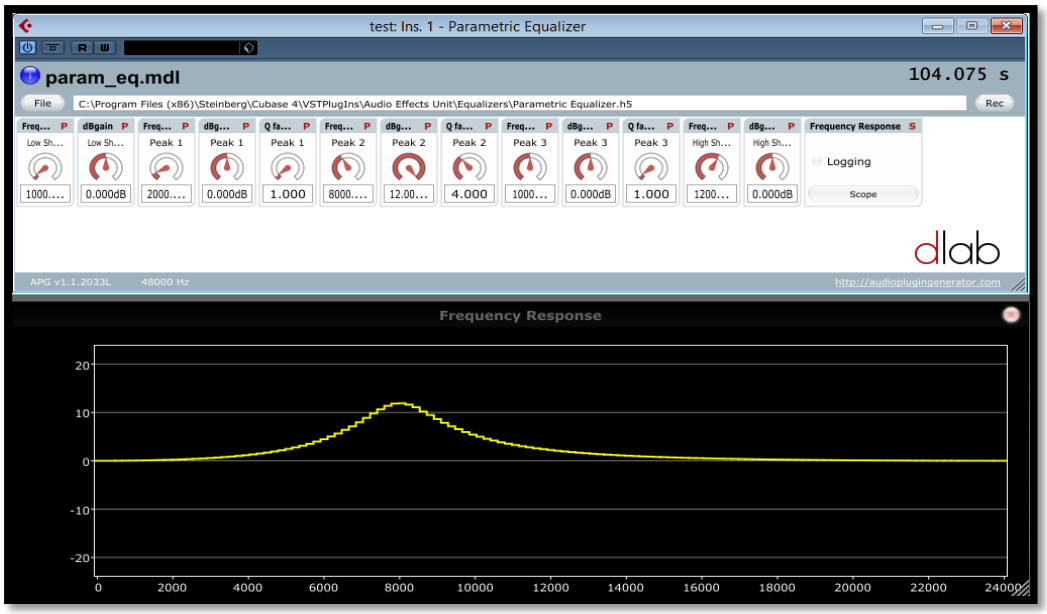

Figura B.1: Ventana de configuración del plugin Parametric Equalizer.

b) **<u>Graphic Equalizer 10, 20 & 30 bands (estructura de ecualizadores *peak* en serie):</u>**

- *DBgain* $_{(1\text{-}30)}$ (dB): Control de ganancia para cada una de las bandas sintonizadas a una frecuencia fija. Rango: [-12dB +12dB].
- *QPonderation*: Constante que permite aumentar o reducir el factor de calidad para así configurar el ecualizador con mayor rizado o mayor interacción entre bandas.
- *Frequency Response:* Visualiza la respuesta en frecuencia de la señal en tiempo real, mediante ruido blanco. Rango [0-24kHz].

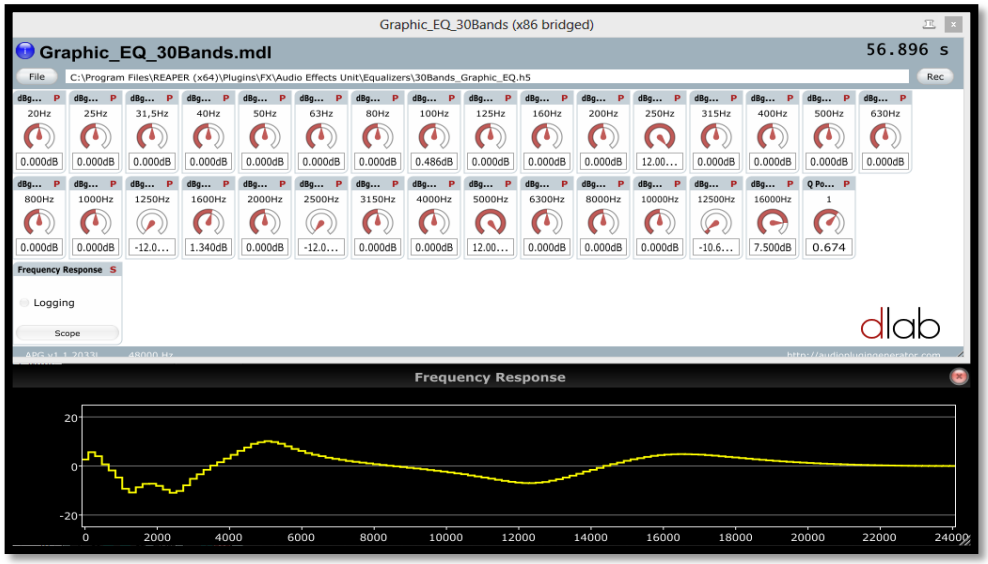

Figura B.2: Ventana de configuración del plugin Graphic Equalizer.

c) **FFT Graphic Equalizer 10, 20 & 30 bands:**

- *DBgain* $_{(1\text{-}30)}$ (dB): Control de ganancia para cada una de las bandas sintonizadas a una frecuencia fija. Rango: [-12dB +12dB].
- *Frequency Response:* Visualiza la respuesta en frecuencia de la señal en tiempo real, mediante ruido blanco. Rango [0-24kHz].
- *Gain*: Control de ganancia de la señal de salida. Rango [0 - 4].

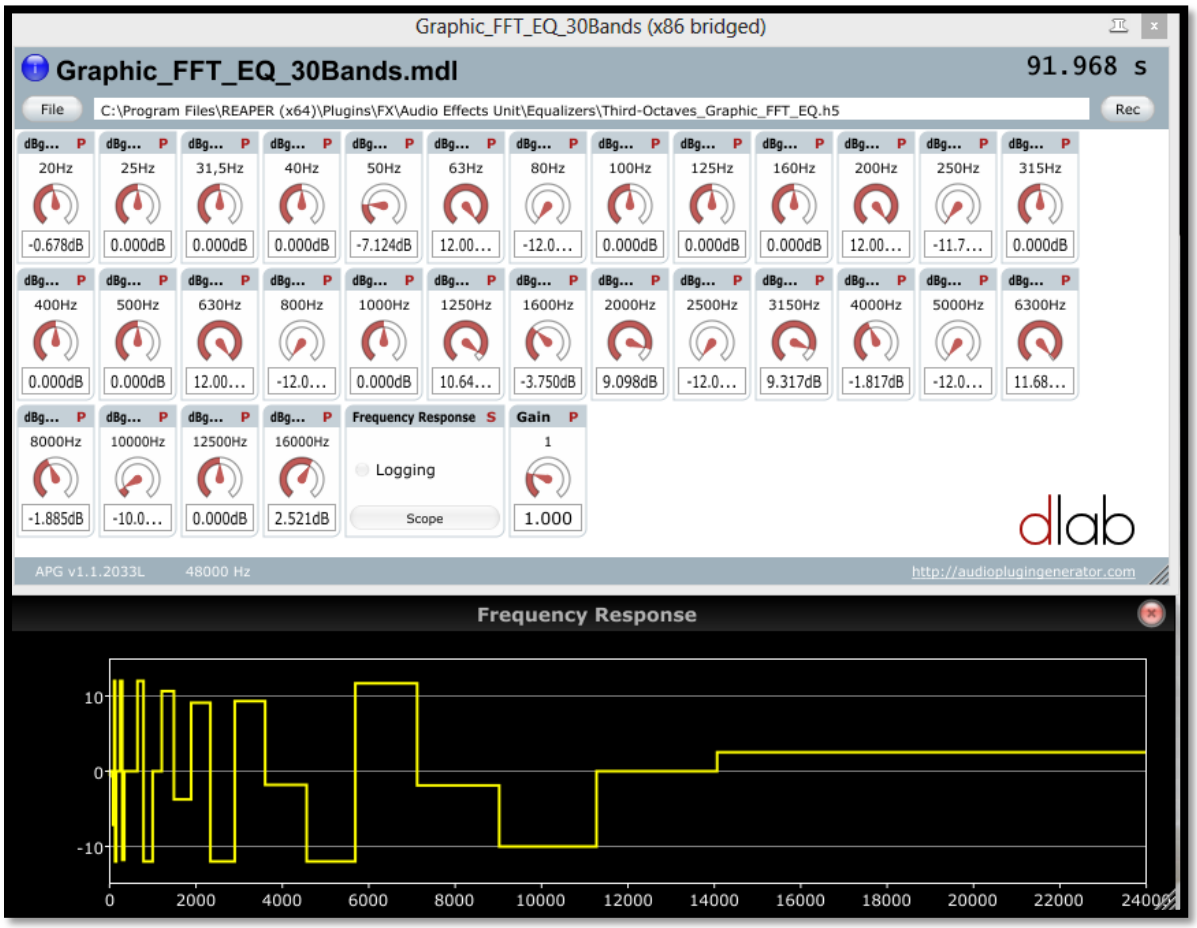

Figura B.3: Ventana de configuración del plugin Graphic FFT Equalizer.

B.2. DYNAMICS

a) **Compressor:**

- *ATime* (ms): Selecciona el tiempo de ataque del compresor. Rango: [0,1ms-100ms].
- *RTime* (ms): Selecciona el tiempo de caída del compresor. Rango: [0,1ms-1000ms].
- *Compressor Treshold* (dB): Selecciona el umbral de nivel a partir del cual se aplica compresión. Rango: [-60dB +0dB].
- *Compressor Rate*: Selecciona el ratio de compresión. Los valores más elevados lo convierten en un compresor-limitador. Rango: [1-10].

- *L Smoother* (samples): Selecciona el número de muestras para suavizar la ganancia. Rango: [2 muestras- 4096 muestras].
- *Gain*: Permite visualizar la rapidez de variación de la ganancia aplicada, para así ajustar el comportamiento del compresor.

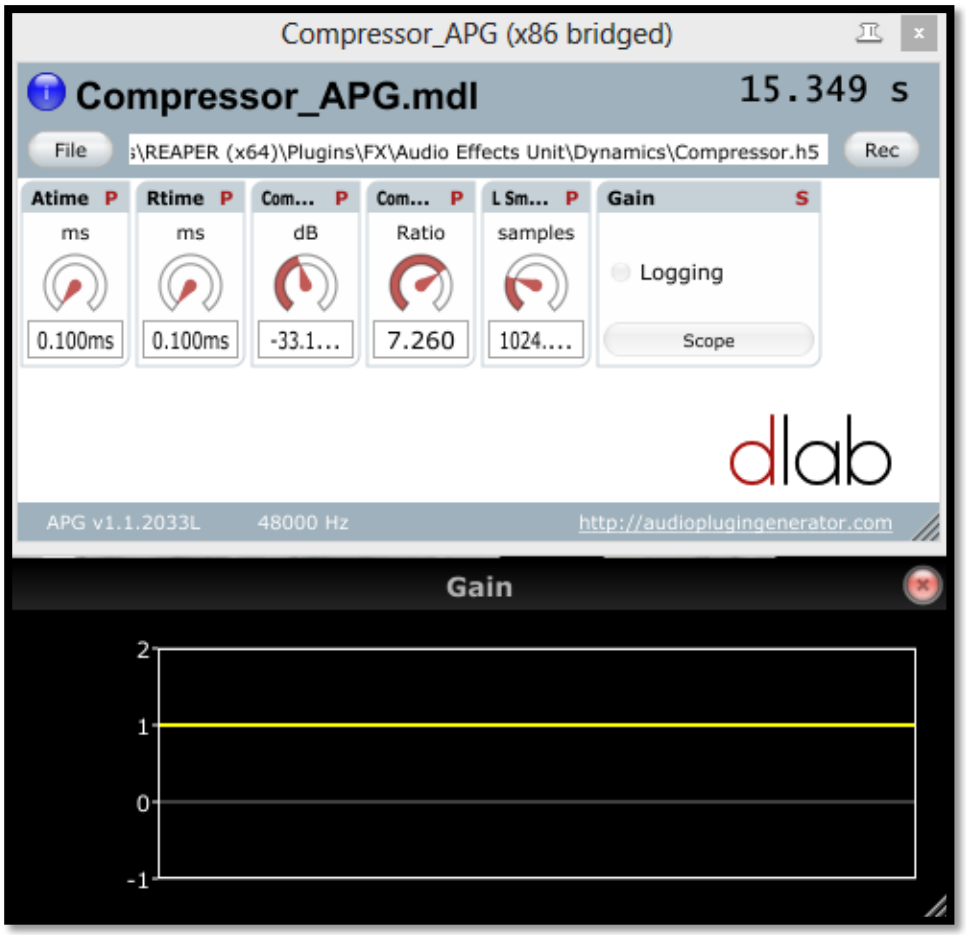

Figura B.4: Ventana de configuración del plugin Compressor.

b) **<u>Expander:</u>**

- *ATime* (ms): Selecciona el tiempo de ataque del expansor. Rango: [0,1ms-100ms].
- *RTime* (ms): Selecciona el tiempo de caída del expansor. Rango: [0,1ms-1000ms].
- *Compressor Treshold* (dB): Selecciona el umbral de nivel a partir del cual se aplica expansión. Rango: [-60dB +0dB].
- *Compressor Rate*: Selecciona el ratio de expansión. Los valores más elevados lo convierten en una puerta de ruido. Rango: [1-10].
- *L Smoother* (samples): Selecciona el número de muestras para suavizar la ganancia. Rango: [2 muestras- 4096 muestras].
- *Gain*: Permite visualizar la rapidez de variación de la ganancia aplicada, para así ajustar el comportamiento del expansor.

Figura B.5: Ventana de configuración del plugin Expander.

B.3. MODULATED DELAY EFFECTS

a) **Chorus:**

- *Delay* (ms): Selecciona la cantidad de *delay* o „profundidad" del *Chorus*. Rango: [1ms-30ms].
- *Blend*: Selecciona la cantidad de señal directa.
- *Feedforward*: Selecciona la cantidad de señal modulada.

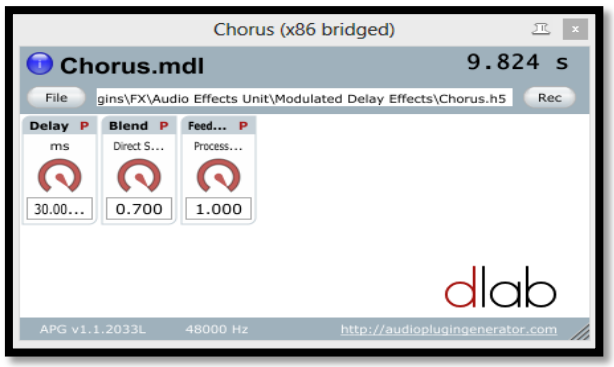

Figura B.6: Ventana de configuración del plugin Chorus.

b) **Echo Delay:**

- *Delay* (ms): Selecciona el *delay* correspondiente al eco aplicado. Rango: [10ms-1000ms].
- *Blend*: Selecciona la cantidad de señal directa. Rango: [0-0.7].
- *Feedforward*: Selecciona la amplitud del eco. Rango: [0-0.7].

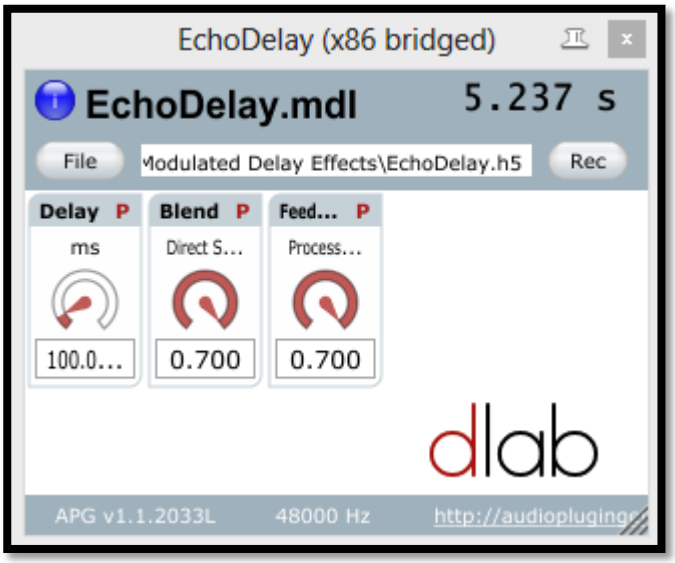

Figura B.7: Ventana de configuración del plugin Echo Delay.

c) **Flanger:**

- *Depth* (ms): Selecciona la cantidad de *delay* o „profundidad" del *Flanger*. Rango: [1ms-15ms].
- *Sine Frequency*: Selecciona la frecuencia de oscilación de la señal sinusoidal moduladora. Rango: [0,1Hz-1Hz].
- *Blend*: Selecciona la cantidad de señal directa. Rango: [0-0.7].
- *Feedforward*: Selecciona la cantidad de señal modulada. Rango: [0-0.7].

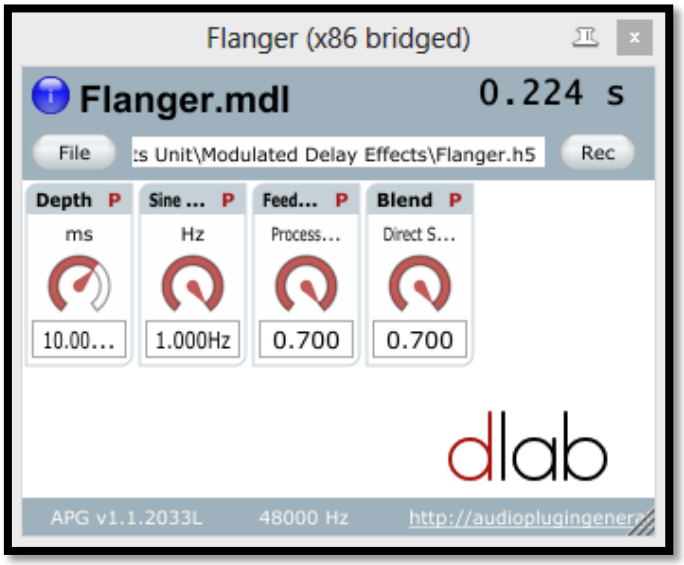

Figura B.8: Ventana de configuración del plugin Flanger.

d) **Vibrato:**

- *Depth* (ms)*:* Selecciona la cantidad de *delay* o „profundidad" del *Vibrato*. Rango: [1ms-10ms].
- *Sine Frequency* (Hz)*:* Selecciona la frecuencia de oscilación de la señal sinusoidal moduladora. Rango: [0,1Hz-14Hz].
- *FeedForward:* Selecciona la cantidad de señal modulada. Rango: [0-1].

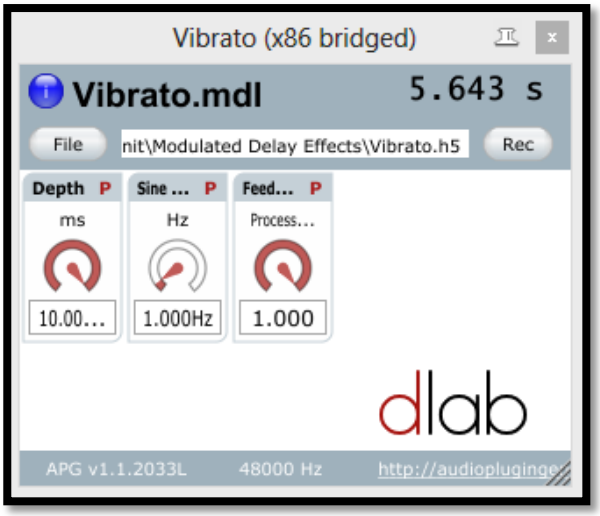

Figura B.9: Ventana de configuración del plugin Vibrato.

B.4. SCHROEDER'S REVERB

- *Reverb Time* (s)*:* Selecciona el tiempo de reverberación deseado. Rango: [0s-10s].
- *Dry:* Selecciona la cantidad de señal directa o „sequedad" del efecto. Rango: [0-1].
- *Wet:* Selecciona la cantidad de señal reverberante. Rango: [0-1].

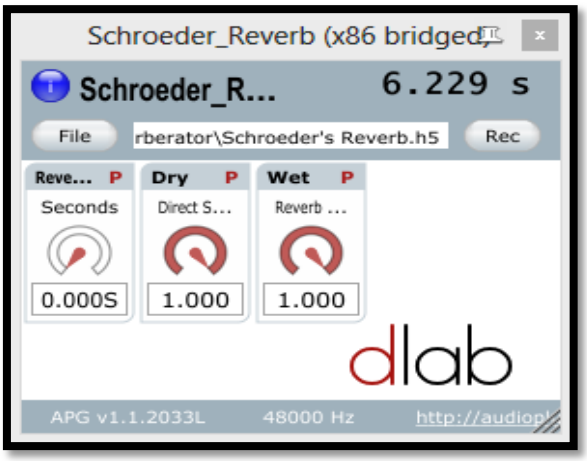

Figura B.10: Ventana de configuración del plugin Schroeder's Reverb.

REFERENCIAS

LIBROS & ARTÍCULOS

[1] Zölder, Udo. *"DAFX: Digital Audio Effects"*. England: John Wiley & Sons (2002). 0-471-49078-4 (Hardback); 0-470-84604-6 (Electronic).

[2] De Götzen, Amalia; Bernardini, Nicola; Arfib, Daniel. "*Traditional implementations of a phase-vocoder: The trick of the trade"*. Proceedings of the COST G-6 Conference on Digital Audio Effects. Verona, Italy (December 7-9,2000).

[3] J.L.Flanagan and R.M. Golden, "*The Phase vocoder"*. The Bell System Technical Journal, 45**,** 8, 1493-1509 (october 1966).

[4] Dolson, Mark, "*The Phase-Vocoder: a tutorial*". Computer Music Journal, 10**,** 4, 14-27, The MIT Press, Cambridge, MA (1986).

[5] Datorro, Jon. *"Effect:Design: Delay-line Modulation and Chorus"*. CCRMA, Stanford University. Stanford, USA (1997).

[6] V. Oppenheim, Alan; W. Schafer, Ronald; R. Buck, John. *"Discrete-Time Signal Processing"*. New Jersey: Prentice Hall (1998).

[7] M. R. Schroeder. *"Natural Sounding Artificial Reverberation"*. Bell Telephone Laboratories, Inc. Murray Hill, New Jersey: 13th Annual Meeting (october 9-13,1961).

[8] J. Orfanidis, Sophocles. *"Introduction to Signal Processing"*. Rutgers University (2010).

[9] J. G. Proakis and D. G. Manolakis.*" Introduction to Digital Signal Processing"*, 2nd ed. Macmillan, New York (1988).

[10] R.E. Crochiere. *"A weighted overlap-add method of short-time Fourier analysis/synthesis"*. IEEE Transactions on Acoustics, Speech, and Signal Processing, 281(1):99-102 (1980).

[11] Xavier Nsabimana, Francois; Zölder, Udo. *"Real-Time implementation of a 1/3-octave audio equalizer. Simulink model using TMS320C6416 DSP"*. Hamburg, Germany: Helmut-Schmidt University & University of the Federal Armed Forces. Department of Signal Processing and Communications.

[12] Zölder, Udo. *"Digital Audio Signal Processing. Second Edition"*. Hamburg, Germany: John Wiley & Sons (2008). Helmut-Schmidt University.

[13] Miller, Ray; Jeffs, Rick; Bohn, Dennis. *"Perfect Q. The next step in Graphic EQ Design"*. Rane Corporation USA: RaneNote 154 (2005).

[14] BS EN, ISO 266:1997. *"Acoustics - Preferred frequencies"*. 15 November 1997.

RECURSOS ONLINE

[15] Dlab GmbH. *"User Guide Audio Plugin Generator"*. Winterthur, Germany. < http://www.dlab.ch>.

[16] Mathworks, Inc. *"Simulink Software description, tutorials & examples"*. <*http://www.mathworks.es/products/simulink/examples.html*>.

[17] Mathworks, Inc. *"Simulink DSP System Toolbox description & examples"*. <*http://www.mathworks.es/products/dsp-system/code-examples.html*>.

[18] Bristow Johnson, Robert. *"Cookbook formulae for audio EQ biquad filter coefficients"*.
<http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt>.